SERIA

INFORMATICA

RYURICK MARIUS HRISTEV

introducere în:

PROLOG

un limbaj al inteligenței artificiale

ZX SPECTRUM
HC
COBRA
TIM-S
CIP
JET

EDITURA APH · BUCUREȘTI 1992

SERIA
INFORMATICA

RYURICK MARIUS HRISTEV

introducere în

PROLOG

un limbaj al inteligenţei artificiale

pentru

ZX SPECTRUM	TIM – S
HC	CIP
COBRA	JET

EDITURA APH A-P. BUCUREŞTI 1992

Fiind o „*Introducere* în PROLOG”, cartea se adresează tuturor celor care doresc să se inițieze în acest limbaj de programare atît de deosebit de celelalte.

Cartea realizează o introducere treptată în conceptele de bază ale limbajului pînă la elaborarea unei aplicații complexe (derivarea formală a funcțiilor de o variabilă) în ultimul capitol, în care cititorul poate sesiza puterea deosebită a acestui limbaj de programare, singurul în care „un matematician poate programa în mod elegant”.

Cartea cuprinde numeroase aplicații de uz general, din domeniul matematicii și al procesării textelor, care pot fi utilizate și dezvoltate în programele proprii.

AUTORUL

str. Cap. Preda nr. 12, sect. 5, 76437 București 69,
tel. 80.20.30, 80.93.97, 80.74.77

Redactor, tehnoredactor, coperta: **R. M. Hristev**

Bun de tipar 2.VI.1992. Apărut 1992. Format 70 × 100/16.
Coli de tipar: 6.

Tiparul executat la Tipografia „UNIVERSUL — SA”
cd. 291/1992

PREFAȚA

PROLOG — un limbaj de programare al inteligenței artificiale. Alături de **Lisp** și împreună cu acesta formează o categorie aparte în lumea limbajelor de programare datorită modului cu totul deosebit de a rezolva problemele. În **PROLOG** problemele sînt mai degrabă descrise decît rezolvate, rezolvarea căzînd în seama compilatorului (interpretorului) prin metode specifice.

Întrucît **PROLOG** prezintă facilități superioare Lisp-ului de programare a problemelor, el a fost ales pentru dezvoltarea sistemelor expert destinate să ruleze pe calculatoarele de generația a V-a, de către experți japonezi care lucrează la aceste proiecte.

PROLOG a fost dezvoltat la începutul anilor '70 în Marsilia (Franța) de către Colmeraurer și echipa sa. **PROLOG** se bazează pe calculul logic formal (este un demonstrator automat de teoreme) astfel încît parcurgerea cu atenție a primului capitol este importantă pentru înțelegerea aprofundată a conceptelor de bază ale limbajului.

Deoarece o practică pe un calculator ușurează mult asimilarea conceptelor, relațiile (exemplele) dezvoltate au fost elaborate sub **Micro-PROLOG** produs de firma LPA Ltd disponibil pe calculatoare compatibile ZX Spectrum și pe cele care lucrează sub sistemul de operare CP/M ușor accesibile cititorilor.

Observațiile din carte se referă la modul concret de operare pe calculatoarele compatibile ZX Spectrum.

Relațiile definite și scrise cu litere **aldine** (grase) pot fi utilizate direct (fără modificări) sub sistemul Micro-PROLOG.

Cuvintele scrise cu litere *cursive* (italice) trebuie înlocuite cu ceea ce reprezintă ele. Exemple: *număr* se înlocuiește cu orice număr, *nume-relație* se înlocuiește cu un cuvînt care reprezintă numele unei relații etc.

CALCUL PROPOZIŢIONAL

1. Propoziţii

Definiţie : Propoziţia este un enunţ care poate fi adevărat sau fals.
Oricărei propoziţii i se asociază o valoare de adevăr :
1 dacă propoziţia este adevărată
0 dacă propoziţia este falsă

2. Operatori logici

Cu ajutorul operatorilor logici din propoziţiile date se pot forma noi propoziţii a căror valoare de adevăr depinde numai de valoarea de adevăr a propoziţiilor date.

Operatorii logici sînt : $\neg$ negaţia (non ; not)
$\wedge$ conjuncţia (şi ; and)
$\vee$ disjuncţia (sau ; or)
$\Rightarrow$ implicaţia
$\Leftrightarrow$ echivalenţa

Tabelele de adevăr ale operatorilor sînt (A, B sînt propoziţii)

negaţia

A	$\neg$A
1	0
0	1

conjuncţia

A	B	A $\wedge$ B
1	1	1
1	0	0
0	1	0
0	0	0

disjuncţia

A	B	A $\vee$ B
1	1	1
1	0	1
0	1	1
0	0	0

implicaţia

A	B	A $\Rightarrow$ B
1	1	1
1	0	0
0	1	1
0	0	1

echivalenţa

A	B	A $\Leftrightarrow$ B
1	1	1
1	0	0
0	1	0
0	0	1

3. Legile calculului propoziţional

Definiţie : Formulele (variabilele propoziţionale) se formează după 2 reguli :
- propoziţiile sînt formule
- dacă A şi B sînt formule atunci $(\neg A)$, $(A \wedge B)$, $(A \vee B)$, $(A \Rightarrow B)$ şi $(A \Leftrightarrow B)$ sînt formule

Observaţie : În general parantezele nu se notează ; atunci ordinea de prioritate a operatorilor este : $\neg$, $\wedge$, $\vee$, $\Rightarrow$, $\Leftrightarrow$ la evaluare de la stînga la dreapta.

O formulă a calculului propoziţional s.n. lege (tautologie, formulă identic adevărată) dacă orice valoare de adevăr ar avea variabilele propoziţionale care intră în compunerea sa, valoarea de adevăr a propoziţiei obţinute este 1.

1) $A \vee \neg A$ (legea terţului exclus)

A	$\neg A$	$A \vee \neg A$
1	0	1
0	1	1

2) $\neg(A \Rightarrow B) \Leftrightarrow A \vee \neg B$ (legea negării implicaţiei)

A	B	$A \Rightarrow B$	$\neg B$	$(A \Rightarrow B)$	$A \wedge \neg B$	$\neg(A \Rightarrow B) \Leftrightarrow A \wedge \neg B$
1	1	1	0	0	0	1
1	0	0	1	1	1	1
0	1	1	0	0	0	1
0	0	1	1	0	0	1

3) $(A \Rightarrow B) \wedge (B \Rightarrow C) \Rightarrow (A \Rightarrow C)$ (legea silogismului)

A	B	C	$A \Rightarrow B$	$B \Rightarrow C$	$(A \Rightarrow B) \wedge (B \Rightarrow C)$	$A \Rightarrow C$	$(A \Rightarrow B) \wedge (B \Rightarrow C) \Rightarrow (A \Rightarrow C)$
1	1	1	1	1	1	1	1
1	1	0	1	0	0	0	1
1	0	1	0	1	0	1	1
1	0	0	0	1	0	0	1
0	1	1	1	1	1	1	1
0	1	0	1	0	0	1	1
0	0	1	0	1	0	1	1
0	0	0	0	1	0	1	1

4) $A \Leftrightarrow A$ (legea de reflexivitate)

A	$A \Leftrightarrow A$
1	1
0	1

5) $A \wedge A \Leftrightarrow A$

$A \vee A \Leftrightarrow A$ (legile de idempotență)

A	$A \wedge A$	$A \wedge A \Leftrightarrow A$
1	1	1
0	0	1

A	$A \vee A$	$A \vee A \Leftrightarrow A$
1	1	1
0	0	1

6) $A \wedge B \Leftrightarrow B \wedge A$

$A \vee B \Leftrightarrow B \vee A$ (legile de comutativitate)

A	B	$A \wedge B$	$B \wedge A$	$A \wedge B \Leftrightarrow B \wedge A$
1	1	1	1	1
1	0	0	0	1
0	1	0	0	1
0	0	0	0	1

A	B	$A \vee B$	$B \vee A$	$A \vee B \Leftrightarrow B \vee A$
1	1	1	1	1
1	0	1	1	1
0	1	1	1	1
0	0	0	0	1

7) $A \wedge (B \wedge C) \Leftrightarrow (A \wedge B) \wedge C$

$A \vee (B \vee C) \Leftrightarrow (A \vee B) \vee C$ (legile de asociativitate)

A	B	C	$A \wedge B$	$B \wedge C$	$A \wedge (B \wedge C)$	$(A \wedge B) \wedge C$	$A \wedge (B \wedge C) \Leftrightarrow (A \wedge B) \wedge C$
1	1	1	1	1	1	1	1
1	1	0	1	0	0	0	1
1	0	1	0	0	0	0	1
1	0	0	0	0	0	0	1
0	1	1	0	1	0	0	1
0	1	0	0	0	0	0	1
0	0	1	0	0	0	0	1
0	0	0	0	0	0	0	1

A	B	C	$A \vee B$	$B \vee C$	$A \vee (B \vee C)$	$(A \vee B) \vee C$	$A \wedge (B \vee C) \Leftrightarrow (A \vee B) \vee C$
1	1	1	1	1	1	1	1
1	1	0	1	1	1	1	1
1	0	1	1	1	1	1	1
1	0	0	1	0	1	1	1
0	1	1	1	1	1	1	1
0	1	0	1	1	1	1	1
0	0	1	0	1	1	1	1
0	0	0	0	0	0	0	1

8) $A \wedge (B \vee C) \Leftrightarrow (A \wedge B) \vee (A \wedge C)$

$A \vee (B \wedge C) \Leftrightarrow (A \vee B) \wedge (A \vee C)$ (legile de distributivitate)

A	B	C	$B \vee C$	$A \wedge (B \vee C)$	$A \wedge B$	$A \wedge C$	$(A \wedge B) \vee (A \wedge C)$	$A \wedge (B \vee C) \Leftrightarrow (A \wedge B) \vee (A \wedge C)$
1	1	1	1	1	1	1	1	1
1	1	0	1	1	1	0	1	1
1	0	1	1	1	0	1	1	1
1	0	0	0	0	0	0	0	1
0	1	1	1	0	0	0	0	1
0	1	0	1	0	0	0	0	1
0	0	1	1	0	0	0	0	1
0	0	0	0	0	0	0	0	1

A	B	C	$B \wedge C$	$A \vee (B \wedge C)$	$A \vee B$	$A \vee C$	$(A \vee B) \wedge (A \vee C)$	$A \vee (B \wedge C) \Leftrightarrow (A \vee B) \wedge (A \vee C)$
1	1	1	1	1	1	1	1	1
1	1	0	0	1	1	1	1	1
1	0	1	0	1	1	1	1	1
1	0	0	0	1	1	1	1	1
0	1	1	1	1	1	1	1	1
0	1	0	0	0	1	0	0	1
0	0	1	0	0	0	1	0	1
0	0	0	0	0	0	0	0	1

9) $\neg\neg A \Leftrightarrow A$ (legea dublei negații)

A	$\neg A$	$\neg\neg A$	$\neg\neg A \Leftrightarrow A$
1	0	1	1
0	1	0	1

10) $(A \Leftrightarrow B) \Leftrightarrow (B \Leftrightarrow A)$

A	B	$A \Leftrightarrow B$	$B \Leftrightarrow A$	$(A \Leftrightarrow B) \Leftrightarrow (B \Leftrightarrow A)$
1	1	1	1	1
1	0	0	0	1
0	1	0	0	1
0	0	1	1	1

11) $(A \Rightarrow B) \Leftrightarrow (\neg B \Rightarrow \neg A)$

A	B	$\neg A$	$\neg B$	$A \Rightarrow B$	$\neg B \Rightarrow \neg A$	$(A \Rightarrow B) \Leftrightarrow (\neg B \Rightarrow \neg A)$
1	1	0	0	1	1	1
1	0	0	1	0	0	1
0	1	1	0	1	1	1
0	0	1	1	1	1	1

12) $(A \Leftrightarrow B) \Leftrightarrow (\neg B \Leftrightarrow \neg A)$

A	B	$\neg A$	$\neg B$	$A \Leftrightarrow B$	$\neg B \Leftrightarrow \neg A$	$(A \Leftrightarrow B) \Leftrightarrow (\neg B \Leftrightarrow \neg A)$
1	1	0	0	1	1	1
1	0	0	1	0	0	1
0	1	1	0	0	0	1
0	0	1	1	1	1	1

13) $(A \Leftrightarrow B) \Leftrightarrow (A \Rightarrow B) \wedge (B \Rightarrow A)$

A	B	$A \Leftrightarrow B$	$A \Rightarrow B$	$B \Rightarrow A$	$(A \Rightarrow B) \wedge (B \Rightarrow A)$	$(A \Leftrightarrow B) \Leftrightarrow (A \Rightarrow B) \wedge (B \Rightarrow A)$
1	1	1	1	1	1	1
1	0	0	0	1	0	1
0	1	0	1	0	0	1
0	0	1	1	1	1	1

14) $A \wedge (A \Rightarrow B) \Rightarrow B$

A	B	$A \Rightarrow B$	$A \wedge (A \Rightarrow B)$	$A \wedge (A \Rightarrow B) \Rightarrow B$
1	1	1	1	1
1	0	0	0	1
0	1	1	0	1
0	0	1	0	1

1) $(A \Leftrightarrow B) \wedge (B \Leftrightarrow C) \Rightarrow (A \Leftrightarrow C)$

A	B	C	$A \Leftrightarrow B$	$B \Leftrightarrow C$	$A \Leftrightarrow C$	$(A \Leftrightarrow B) \wedge (B \Leftrightarrow C)$	$(A \Leftrightarrow B) \wedge (B \Leftrightarrow C) \Rightarrow (A \Leftrightarrow C)$
1	1	1	1	1	1	1	1
1	1	0	1	0	0	0	1
1	0	1	0	0	1	0	1
1	0	0	0	1	0	0	1
0	1	1	0	1	0	0	1
0	1	0	0	0	1	0	1
0	0	1	1	0	0	0	1
0	0	0	1	1	1	1	1

16) $(\neg A \Rightarrow B) \wedge (\neg A \Rightarrow \neg B) \Rightarrow A$

A	B	$\neg A$	$\neg B$	$\neg A \Rightarrow B$	$\neg A \Rightarrow \neg B$	$(\neg A \Rightarrow B) \wedge (\neg A \Rightarrow \neg B)$	$(\neg A \Rightarrow B) \wedge (\neg A \Rightarrow \neg B) \Rightarrow A$
1	1	0	0	1	1	1	1
1	0	0	1	1	1	1	1
0	1	1	0	1	0	0	1
0	0	1	1	0	1	0	1

17) $\neg(A \wedge B) \Leftrightarrow \neg A \vee \neg B$

$\neg(A \vee B) \Leftrightarrow \neg A \wedge \neg B$ (legile lui de Morgan)

A	B	$\neg A$	$\neg B$	$A \wedge B$	$\neg(A \wedge B)$	$\neg A \vee \neg B$	$\neg(A \wedge B) \Leftrightarrow \neg A \vee \neg B)$
1	1	0	0	1	0	0	1
1	0	0	1	0	1	1	1
0	1	1	0	0	1	1	1
0	0	1	1	0	1		

A	B	¬A	¬B	A ∨ B	¬(A ∨ B)	¬A ∧ ¬B	¬(A ∨ B) ⇔ ¬A ∧ ¬B
1	1	0	0	1	0	0	1
1	0	0	1	1	0	0	1
0	1	1	0	1	0	0	1
0	0	1	1	0	1	1	1

18) $(A \Rightarrow B) \wedge (B \Rightarrow C) \Rightarrow (A \vee B \Rightarrow C)$

A	B	C	A⇒C	B⇒C	(A⇒C) ∧ (B⇒C)	A∨ B	(A ∨ B)⇒C	(A⇒C) ∧ (B⇒C)⇒(A ∨ B⇒C)
1	1	1	1	1	1	1	1	1
1	1	0	0	0	0	1	0	1
1	0	1	1	1	1	1	1	1
1	0	0	0	1	0	1	1	1
0	1	1	1	1	1	1	1	1
0	1	0	1	0	0	1	0	1
0	0	1	1	1	1	0	1	1
0	0	0	1	1	1	0	1	1

4. Predicate

Definiție : Predicatul este un enunț care depinde de una sau mai multe variabile și care are proprietatea că pentru anumite valori ale variabilelor (valori care sînt elemente ale unei mulțimi) devine o propoziție.

Un predicat care depinde de n variabile se numește predicat n-ar.

Observație : Folosind predicate și operatori logici se pot construi alte predicate.

Definiție : Predicatul $B(x_1 \ldots x_n)$ se numește consecință logică a predicatului $A(x_1 \ldots x_n)$; notație : $A(x_1 \ldots x_n) \Rightarrow B(x_1 \ldots x_n)$; dacă pentru orice valori ale variabilelor $x_1, \ldots x_n$ propoziția $A(x_1 \ldots x_n) \Rightarrow B(x_1 \ldots x_n)$ este adevărată.

Predicatele $A(x_1 \ldots x_n)$ și $B(x_1 \ldots x_n)$ se numesc echivalente logic ; notație : $A(x_1 \ldots x_n) \Leftrightarrow B(x_1 \ldots x_n)$; dacă pentru orice valori ale variabilelor $x_1 \ldots x_n$ propoziția $A(x_1 \ldots x_n) \Leftrightarrow B(x_1 \ldots x_n)$ este adevărată.

5. Propoziții universale și existențiale

Definiție : Fie A(x) un predicat unar. Propoziția „pentru oricare valoare permisă a variabilei x, A(x) este o propoziție adevă-

rată" se numește propoziția universală asociată predicatului A(x) notație : $(\forall x)\ A(x)$.

Definiție : Fie A(x) un predicat unar. Propoziția „există cel puțin o valoare x_0 a variabilei x astfel încît $A(x_0)$ să fie o propoziție adevărată" se numește propoziția existențială asociată predicatului A(x) ; notație : $(\exists x)A(x)$ (notație : $\exists !\, x$ există un unic x)

Fie A(x) definit pentru un număr finit de valori ale variabilei x : $(x_1 \ldots x_n)$
Atunci :

$$(\forall x)A(x) \Leftrightarrow A(x_1) \wedge \ldots \wedge A(x_n)$$

$$(\exists x)A(x) \Leftrightarrow A(x_1) \vee \ldots \vee A(x_n)$$

Conform legilor de Morgan rezultă regulile de negație :

$$\neg(\forall x)A(x) \Leftrightarrow \neg A(x_1) \vee \ldots \vee A(x_n) \Leftrightarrow (\exists x)(\neg A(x))$$

$$\neg(\exists x)A(x) \Leftrightarrow \neg A(x_1) \wedge \ldots \wedge A(x_n) \Leftrightarrow (\forall x)(\neg A(x))$$

Observație : Regulile de negație stabilite sînt valabile și pentru o mulțime infinită de valori pentru variabila x.

Fie A(x, y) un predicat binar

atunci $(\forall x)A(x, y)$ este predicat unar care depinde de variabila y deci se pot forma propozițiile $(\forall y)(\forall x)A(x, y)$ și $(\exists y)(\forall x)A(x, y)$

analog $\exists(x)A(x, y)$ este predicat unar care depinde de variabila y deci se pot forma propozițiile $(\forall y)(\exists x)A(x, y)$ și $(\exists y)(\exists x)A(x, y)$

Regulile de negație :

$$\neg((\forall y)(\forall x)A(x, y)) \Leftrightarrow (\exists y)(\neg(\forall x)A(x, y)) \Leftrightarrow (\exists y)(\exists x)(\neg A(x, y))$$

$$\neg((\exists y)(\forall x)A(x, y)) \Leftrightarrow (\forall y)(\neg(\forall x)A(x, y)) \Leftrightarrow (\forall y)(\exists x)(\neg A(x, y))$$

$$\neg(\forall y)(\exists x)A(x, y)) \Leftrightarrow (\exists y)(\neg(\exists x)A(x, y)) \Leftrightarrow (\exists y)(\forall x)(\neg A(x, y))$$

$$\neg((\exists y)(\exists x)A(x, y)) \Leftrightarrow (\forall v)(\neg(\exists x)A(x, y)) \Leftrightarrow (\forall y)(\forall x)(\neg A(x, y))$$

BAZE DE DATE

1. Introducere

Limbajul de programare PROLOG (PROgramming in LOGic) a fost dezvoltat și implementat pe calculatoare pentru prima oară în 1972 în Marsilia (Franța) de către Colmerauer și echipa sa. PROLOG se bazează pe calculul logic formal care descrie și reprezintă raționamentul uman. Ca atare limbajul PROLOG diferă esențial de toate celelalte limbaje de programare (Basic, Pascal, C) cu excepția Lisp cu care prezintă unele asemănări, fiind un limbaj de tip declarativ (Basic, Pascal, C sînt limbaje imperative).

Un program PROLOG este alcătuit din propoziții care definesc relații dintre diferite tipuri de obiecte care pot fi : numere, mulțimi (liste), relații, cuvinte etc., în general : obiecte. În PROLOG nu se face distincție între baza (colecția) de date și program, între regăsirea informațiilor și calcul efectiv.

Pentru a ușura înțelegerea conceptelor și sintaxei PROLOG se va folosi o sintaxă simplificată, urmînd ca în penultimul capitol să se treacă la sintaxa PROLOG standard.

Observație : Această sintaxă simplificată este disponibilă pe calculatoarele din clasa Spectrum. Pentru aceasta : după încărcarea Micro PROLOG*) se tastează literă cu literă : **L O A D ␣ S I M P L E** (blank-ul este separator). SIMPLE este un program scris în PROLOG care cuprinde 26 blocuri scurte de lungime 104H bytes fiecare. Dacă unul din blocuri nu se încarcă sau s-a încărcat cu eroare se reia încărcarea de la blocul respectiv.

2. Alcătuirea bazelor de date

A) Introducerea propozițiilor

O propoziție este de forma :

nume-obiect␣nume-relație␣nume-obiect

Observație : *"&."* este prompt-ul prin care Micro PROLOG semnalizează că este în așteptarea unei comenzi.

---

*) Micro PROLOG este produs înregistrat al firmei Logic Programming Associates Ltd.

Propozițiile se introduc folosind comanda add :

&. a d d (*propoziție*)

Exemple de propoziții :

R o m â n i a_a r e — c a p i t a l a_B u c u r e ș t i

F r a n ț a_a r e — c a p i t a l a_P a r i s

Observație : Corecturile de tastare se fac folosind DELETE și cursorul. După terminarea tastării unei comenzi se apasă ENTER. În cazul existenței unor greșeli apare un mesaj de eroare. Prompt-ul "*număr* ." reprezintă numărul parantezelor drepte necesare pentru a termina în mod corect o propoziție; apare de asemenea și cînd o propoziție se întinde pe mai multe linii.

Relațiile între obiecte sînt :

binare : *nume—obiect_nume—relație_nume—obiect*

unare : *nume—obiect_nume—proprietate*

Exemple : **carte_făcut—din_hîrtie**

scaun_făcut—din_lemn

stilou_instrument—de—scris

pix_instrument—de—scris

creion_instrument—de—scris

Există 3 forme de a scrie tipurile de relații între obiecte :

forma post-fixată : *nume—obiect_nume—proprietate*

forma prefixată : *nnme—relație (nume—obiect... nume—obiect)*

forma infixată : *nume—obiect_nume—relație_nume—obiect*

Observație : Spațiul (blank-ul notat _ sau ␢), parantezele "(" și ")" și liniile noi generate prin apăsarea ENTER sînt separatori de cuvinte. (În general toate caracterele cu excepția cifrelor, literelor mari și mici, "—" minus și "_" sublinierea sînt separatori).

Numerele, "—" și literele se consideră caractere alfabetice.
Numele-obiect sînt constante alfanumerice. Atenție : un *nume-obiect* nu poate începe cu o cifră. Dacă o constantă trebuie să conțină caractere care sînt separatori, atunci întreaga constantă va fi scrisă între ghilimele.
Propozițiile pot fi introduse și prin comanda "**accept**"

&. **a c c e p t** *nume—relație*

în acest caz prompt-ul devine "*nume—relație* .", apoi se introduc listele de argumente, fiecare în paranteze. La terminare se tastează :

nume—relație **. e n d**

și se revine la situația anterioară comenzii **accept**.

B) Listarea, înregistrarea și încărcarea programelor

Listarea programului se face cu comanda „**list**" :

pentru listarea întregului program : & . **l i s t a l l**

pentru listarea unei relații : & . **l i s t** *nume—relație*

Înregistrarea pe bandă magnetică a programelor se face cu comanda **save :**

& . **s a v e** *nume—program*

Încărcarea programelor de pe bandă magnetică se face cu comanda **load**:

&. **load** *nume—program*

C) Editarea

Pentru a șterge o propoziție din baza de date se folosește comanda „**delete**" (care este opusă comenzii „**add**"):

&. **delete** (*propoziție*)

sau prin referire la poziția propoziției în lista *nume—relație* :

&. **delete** *nume—relație număr*

La folosirea comenzii „**add**" noua propoziție este listată la sfîrșit deoarece a fost introdusă ultima. O altă folosire a comenzii „**add**" este

&. **add** *număr* (*propoziție*)

care adaugă *propoziția* în listă în poziția indicată de *număr*
Comanda „**kill**":

&. **kill** *nume—relație* — șterge toate propozițiile în care apare *nume—relație*

&. **kill all** — șterge tot programul din zona de lucru

O propoziție poate fi modificată prin folosirea comenzii edit:

&. edit. *nume-relație număr*

afișează propoziția respectivă pentru a fi editată
Comanda „**NEW**" este o primitivă PROLOG care resetează compilatorul (interpretorul) PROLOG (deci șterge toate programele)

&. **NEW.**

Orice comandă PROLOG are cel puțin un argument. Pentru „**NEW**" poate fi folosit orice argument (de exemplu ".")

Observație: La înregistrare pe bandă magnetică a unui program nume—program trebuie să difere de numele oricărei relații sau comenzi. În caz contrar apare mesajul „file error" și operația este abandonată.

3. Interogarea bazelor de date

A) Variabile

Literele x, y, z, X, Y, Z, **x1**, **y1**, **z1**, **X1**, **Y1**, **Z1** ... sînt variabile PROLOG

Atenție: **x1**, **x2**, ... nu sînt variabile diferite de **x01**, **x02**, ... sau **x001**, **x002** ... și numai ultimile 2 cifre sînt considerate semnificative.

B) Confirmări

Întrebarea „is" este de forma:

&. **is** (*condiție*)

[*condiție* este un predicat]

unde *condiție* este o propoziție în care **unul** sau mai multe "*nume—obiect*" pot fi înlocuite cu variabile. PROLOG caută în baza de date dacă *condiția*

este adevărată sau falsă. Dacă este adevărată, atunci răspunsul este "YES", dacă este falsă răspunsul este "NO".
O altă formă a întrebării "is" este :

&. is (*condiție* **and** *condiție* ... **and** *condiție*)

C) Regăsirea informațiilor

Întrebarea "**which**" este de forma :

&. which (*sistem—de—soluții* : *condiție* **and** ... *condiție*)

[*condiție* **and** ... *condiție* este un predicat, adevărat pentru *sistem—de—soluții*].

PROLOG afișează toate soluțiile care satisfac conjuncția de condiții. Fiecare răspuns este de forma *sistemului—de—soluții* în care variabilele sînt înlocuite cu numele obiectelor care satisfac : *condiție* **and** ... *condiție*. Variabilele din *sistemul de soluții* trebuie separate prin spații.
După lista tuturor răspunsurilor posibile (eventual nici unul dacă nu există) se afișează mesajul "**No (more) answers**". Dacă " **:**" lipsește apare un mesaj de eroare.
În locul lui "**and**" poate fi folosit "**&**"
Întrebarea "**all**" este similară "**which**"
Întrebarea "**one**" este de forma :

&. one (*sistem—de—soluții* : *condiție* **&** ... *condiție*)

este asemănătoare cu "**which**" cu diferența că după găsirea fiecărei soluții apare mesajul "**more (y/n) ?**". Dacă se apasă "y" atunci se caută soluția următoare, dacă se apasă "n" căutarea este abandonată.

D) Găsirea relațiilor definite.

Pentru a găsi care nume de relații au fost utilizate se folosește :

&. all (x : x dict) sau

&. list dict

Atenție : Faptul că un *nume—relație* a fost afișat ca răspuns nu garantează existența unei propoziții de definire a relației.

Pentru a afla dacă o relație *R* a fost definită

&. is (*R* defined)

Pentru a găsi toate relațiile pentru care există definiții :

&.all(x : x dict & x defined)

PROLOG conține un set de relații predefinite (primitive). Încercarea de a adăuga la acestea noi propoziții va da mesaj de eroare :

"**Cannot add sentence for *R***" unde *R* este numele relației.

Observație : Același mesaj va fi afișat la încercarea de adăuga o propoziție la relațiile definite în SIMPLE. Definițiile sale sînt protejate în module.

Modulele sînt colecții de definiții de relații care exportă în mod explicit numele unor anumite relații. Numai relațiile exportate pot fi folosite de alte programe iar acestea nu pot modifica definiția lor (sînt protejate).
Pentru a găsi numele relațiilor exportate de SIMPLE :

&. all(x:x reserved)

4. Aritmetica PROLOG

A) Adunarea și scăderea folosind relația SUM

SUM este o primitivă PROLOG

SUM (x y z) este adevărată dacă și numai dacă x + y = z

Verificare:

& . is (SUM *număr număr număr*)

răspunsul va fi: "YES" sau "NO"

Adunare:

& . which (x : SUM (*număr–1 număr–2 x*))

răspunsul va fi *număr* urmat (de "**No (more) answers**"

[(∃) x (SUM (număr–1 număr–2 x))]

Scădere

& . which (x : SUM (x *număr număr*)) sau

& . which (x : SUM (*număr* x *număr*))

[analog]

răspunsul va fi *număr* urmat de "**No (more) answers**"

Restricții: O condiție interogativă pentru SUM poate avea cel mult un argument necunoscut. PROLOG simulează o bază de date pentru SUM și nu poate genera decît o listă finită de răspunsuri. O întrebare cu 2 sau mai multe variabile va genera un mesaj de eroare "**Too many variables**".

Sintaxa numerelor:

a) întreg pozitiv: o secvență de cifre zecimale fără semnul "+" (în caz contrar apare o eroare)

b) întreg negativ: o secvență de cifre zecimale precedată de semnul "–"

c) număr pozitiv în virgulă flotantă: o secvență de cifre zecimale fără semnul "+", care conține un punct zecimal"." opțional poate fi urmat de un exponent care este un număr întreg precedat de "E". Punctul zecimal trebuie să fie precedat de cel puțin o cifră.

d) număr negativ în virgulă flotantă: are aceeași formă ca numărul pozitiv în virgulă flotantă, fiind precedat de semnul "–"

Observație: Numerele în virgulă flotantă pot fi introduse în orice formă dar sînt afișate într-o formă standard:

mantisa între – 10 și 10:

cifră . cifră ... cifră **E** *cifră cifră*

dacă exponentul este 0 atunci acesta nu se mai afișează.

Numerele întregi trebuie să fie în domeniul – 32767, 32767.

Numerele în virgulă flotantă pot avea 8 cifre semnificative, zerourile anterioare nefiind considerate semnificative.

Exponenții trebuie să fie în domeniul – 127, 127.

Dacă răspunsul este prea mic atunci apare un mesaj de eroare "**Arithmetic underflow**"

Dacă răspunsul este prea mare atunci apare un mesaj de eroare "**Arithmetic overflow**".

Atenție: Utilizarea constantelor care pot fi confundate cu cifre impune utilizarea ghilimelelor.

B) Conversia și testarea tipurilor de numere

„INT" este o primitivă PROLOG. Testare:

&. is (*număr* INT)

răspunsul va fi "**YES**" sau "**NO**"

[(∀) x ∈ Z (x INT) sau (∃) x ∈ ℝ (x INT)]

Conversie:

&. which (x : *număr* INT x)

[(∃) x (număr INT x)]

răspunsul va fi *întreg* urmat de "**No (more) answers**"

Restricții: INT este o relație unară sau binară.
Dacă este folosită ca relație unară, argumentul trebuie să fie dat. Dacă este folosită ca relație binară, primul argument trebuie să fie cunoscut iar al doilea o variabilă (necunoscut)

Pentru a testa dacă un număr este partea întreagă a altui număr se poate folosi primitiva PROLOG „EQ":

&. is (*număr* INT x & x EQ *întreg*)

["YES" ⇔ (∃) x ((număr INT x) ∧ (x EQ întreg))]

["NO" ⇒ (∀x) ¬ ((număr INT x) ∧ (x EQ întreg))]

Atenție: „EQ" trebuie pus după „INT"

C) Înmulțirea și împărțirea folosind relația "**TIMES**"

„TIMES" este o primitivă PROLOG

TIMES (x y z) este adevărată dacă și numai dacă x * y = z

Verificare:

&. is (TIMES (*număr număr număr*))

[analog anterior]

răspunsul va fi "**YES**" sau "**NO**"

Verificarea divizibilității:

&. is (TIMES (*număr* x *număr*) & x INT)

răspunsul va fi "**YES**" sau "**NO**"

Înmulțire:

&. which (x : TIMES (*număr număr* x))

răspunsul va fi *număr* urmat de "**No (more) answers**"

Împărțire:

&. which (x : TIMES (x *număr număr*))

răspunsul va fi *număr* urmat de "**No (more) answers**"

Divizibilitate

&. which (x : TIMES (y *număr număr*) &
x INT y)

răspunsul va fi *număr* urmat de "**No (more) answers**"

Atenție la erorile de rotunjire: numerele în virgulă flotantă sînt aproximări ale numerelor reale. O întrebare de tipul "is (**TIMES** (x *număr—1 număr—2*) & **TIMES** (*număr*—1 x *număr*—2)) poate genera răspunsul "**NO**".

Restricţii : La folosirea primitivei **TIMES** poate fi folosită o singură variabilă, dar aceasta poate fi oricare din cele 3 argumente (analog **SUM**).

D) Testarea ordonării folosind relaţia **LESS**

LESS este o primitivă PROLOG care poate fi folosită numai pentru verificare.

LESS (x y) este adevărată dacă x este mai mic decît y în ordinea numerelor dacă x, y sînt numere sau în ordine alfabetică, conform standardului ASCII, dacă sînt nume (cifrele preced majusculele care preced minusculele).

5. Modul de funcţionare PROLOG

Întrebări cu o condiţie :

a) **is** (*P*) unde *P* este o propoziţie fără variabile
PROLOG caută în baza de date comparînd *P* cu fiecare propoziţie Dacă găseşte o propoziţie care coincide cu *P* furnizează răspunsul "**YES**" ; dacă a ajuns la sfîrşitul bazei de date fără a găsi *P* atunci furnizează răspunsul "**NO**".

b) **is** (*P*) unde *P* este o propoziţie cu cel puţin o variabilă în locul unui *nume—obiect* necunoscut. Căutarea se face în acelaşi mod cu diferenţa că la comparaţii PROLOG dă variabilelor valori care sînt *nume—obiect* corespunzătoare din propoziţii.

Atenţie : La întrebarea "**is** (x *nume—relaţie* x)" PROLOG va furniza răspunsul "**NO**" chiar dacă există o propoziţie de forma "*nume—obiect—1 nume—relaţie nume—obiect—2*" dacă *nume—obiect—1* este diferit de *nume—obiect—2*. Dar la întrebarea "**is** (x *nume—relaţie* y)" va furniza răspunsul "**YES**".

PROLOG trebuie să înlocuiască aceleaşi nume de variabile cu aceleaşi nume de obiecte dar nume de variabile diferite pot fi înlocuite cu acelaşi nume de obiect.

Întrebări **which**

which (*V* : *C*) unde *V* este o listă de variabile a căror valori verifică condiţia *C*

PROLOG compară *C* cu fiecare propoziţie din baza de date.

În momentul în care găseşte o comparaţie reuşită afişează valorile care au fost atribuite variabilelor în aceeaşi ordine în care apar variabilele în *V* apoi continuă căutarea pînă la sfîrşitul bazei de date. În momentul în care a ajuns la sfîrşitul bazei de date, indiferent de numărul soluţiilor găsite (acest număr poate fi şi 0) afişează mesajul "**No (more) answers**"

Dacă *C* este o conjuncţie de condiţii, acestea sînt evaluate prin rezolvare pe rînd, în ordine, de la stînga la dreapta.

Observație : Pentru a vedea modul în care PROLOG rezolvă întrebările „which" și „is" se încarcă programul „SIMTRACE" (13 blocuri).

&. **load SIMTRACE** și

&. **is — trace** (*V* : *C*) sau

&. **all — trace** (*V* : *C*)

se apasă "y" sau "n" și ENTER pentru da sau nu

La sfîrșit comanda

&. **kill simtrace — mod**

va șterge „SIMTRACE".

În cazul condițiilor complexe și a bazelor de date mari răspunsul la întrebări necesită timp mai mult. Pentru a micșora timpul de răspuns condițiile mai restrictive (cu soluții mai puține) se pun la început. În acest fel numărul eventualelor soluții scade și deci scade și timpul necesar verificării lor.

REGULI

1. Reguli

Regulile sînt propoziţii condiţionale de forma *P* if *C*.

propoziţie if *condiţie* & ... *condiţie*
[P ⇒ C ∧ ... C]

care se citesc: Propoziţia *P* este adevărată pentru toate soluţiile pentru care condiţia de forma *C* este adevărată.

Regulile pot folosi alte relaţii definite.

Regulile sînt listate în ordinea în care au fost introduse. PROLOG poate schimba numele unor variabile dar nu poate schimba tipul lor.

Exemplu: regulile pentru relaţia "**maxim—dintre**"

```
x maxim—dintre (x x)
y maxim—dintre (x y) if x LESS y
x maxim—dintre (x y) if y LESS x
```

În întrebarea **which** poate fi introdus text care va fi afişat în răspuns.

&. which (x *text* y : *condiţie*)

generează răspunsul "*obiect—x text obiect—y*". Textul nu influenţează găsirea soluţiei.

2. Modul de funcţionare PROLOG

La întrebări de forma:

&. which (*V* : *C* & *C'*...)

PROLOG găseşte soluţii prin metoda back-traking (căutare înapoi). Pentru a găsi toate soluţiile unei conjuncţii de condiţii se procedează în felul următor:

— se caută o soluţie pentru prima condiţie;
— cu aceasta se merge la condiţia următoare şi se verifică dacă este o soluţie şi pentru ea;
— dacă s-a găsit o soluţie şi pentru condiţia următoare se merge mai departe în mod asemănător;
— dacă soluţia nu verifică condiţia respectivă, atunci se caută soluţia următoare pentru condiţia anterioară, după care se repetă procedeul.

Cînd rezolvarea unei condiții implică aplicarea unei reguli PROLOG întrerupe căutarea de propoziții în baza de date și generează o întrebare auxiliară, conform regulii întîlnite. Fiecare răspuns la această întrebare suplimentară este o soluție pentru condiția respectivă.

Pentru a găsi toate soluțiile unei singure condiții se caută toate propozițiile care verifică condiția respectivă la care se adaugă toate soluțiile regulilor implicate în condiție.

Observație : Folosirea „SIMTRACE". În timpul rezolvării, regula care este folosită pentru a încerca găsirea soluțiilor condițiilor puse în întrebare este identificată prin poziția sa în lista propozițiilor pentru relația respectivă. Dacă este găsită o soluție se afișează noile întrebări impuse de condițiile preliminare ale regulii *"noi—întrebări condiții—preliminare"*.
În identificatorul *"(număr—1, număr—2...)"* *număr—1* semnifică poziția condiției sau regulii în întrebarea curentă, restul numerelor fiind o istorie înapoi la întrebarea inițială.

2. Reguli definite recursiv

Pentru relații care nu pot fi definite decît recursiv (adică prin definiții care se referă la relația care se definește) utilizarea regulilor este fundamentală.

Atenție : Orice definiție recursivă trebuie să aibă o parte nerecursivă, care să asigure ieșirea din recursivitate, în caz contrar ea fiind complet circulară. Partea recursivă a definiției trebuie pusă după partea nerecursivă în special dacă definiția este utilizată pentru a găsi toate soluțiile intermediare ale relației.

Pentru relațiile inverse este mai eficient să se utilizeze definiții separate.

Exemple :

1) definirea relației **"mai—mic—egal"**

```
x mai—mic—egal x
x mai—mic—egal y if x LESS y
```

2) definirea relației **"mai—mare—decît"**

```
x mai—mare—decît y if y LESS x
```

3) definirea relației **"mai—mare—egal"**

```
x mai—mare—egal x
x mai—mare—egal y if y LESS x
```

4) definirea relației **"divizibil—cu"**

```
x divizibil—cu y if x INT & y INT &
        TIMES (y z x) & z INT
```

Atenție : Din cauza restricțiilor impuse primitivelor aritmetice, aceste relații pot fi folosite doar pentru verificare.

Definirea relației "**factorial**"

x factorial y este adevărată dacă și numai dacă

$y = x! = 1 \cdot 2 \cdot \ldots (x-1) \cdot x$

```
1 factorial 1
x factorial y if
  x INT & 1 LESS x &
  SUM (z 1 x)
  z factorial X &
  TIMES (x X y)
  [x = 1 ⇒ x ! = 1
   x ! = y ⇐
      ((x INT) ∧ (1 < x) ∧
      (z = x − 1) ∧
       z! = X) ∧
      (y = x . X))]
```

Această definiție se citește astfel:

Pentru a găsi un y astfel încît $y = x!$ pentru un x dat
- dacă $x = 1$, atunci $y = 1$ (partea nerecursivă) sau
- verifică că x este întreg și $1 < x$
- scade 1 din x pentru a obține z
- găsește z astfel încît $X = (z)!$
- înmulțește X cu x pentru a obține y

Pentru a găsi factorialul unui întreg se folosește o întrebare de forma:

&. which (x : *întreg* **factorial** x)

Primul argument al relației "**factorial**" trebuie specificat datorită restricției pentru primitivele **INT** și **LESS**.

Pentru a verifica relația factorial între 2 întregi se folosește o întrebare de forma:

&. is (*întreg—1* **factorial** *întreg—2*)

Definirea relației "**între**"

x între (y z) adevărată dacă și numai dacă $y \leqslant x < z$

(x, y, z întregi)

```
x între (x z) if x INT & z INT & x LESS z
x între (y z) if
      y INT & z INT &
      SUM (y 1 X) &
      X LESS z & x între (X z)
```

Această definiție se citește astfel:

x între (y z) date dacă (întîi se face verificarea y, z INT)
x = y și x L E S S z adevărat (partea nerecursivă) sau
adaugă 1 la y pentru a obține X apoi
dacă X L E S S z găsește un x astfel încît x între (X, z)

Definirea relației factorial inverse "**factorial—al**":

x factorial—al y este adevărată dacă și numai dacă x = y!

x f a c t o r i a l — a l y i f y î n t r e (1 x) &
y f a c t o r i a l x

[(y î n t r e (1 x)) ∧ (y f a c t o r i a l x) ⇒ (x factorial—al y)]

Condiția suplimentară "y **între** (1 x)" este logic redundantă dar ea are rolul de a da valori lui y înainte de a evalua (calcula) "y factorial x". Definiția poate fi folosită numai pentru a-l găsi pe y nu pe x în caz contrar evaluarea expresiei "y **î n t r e** (1 x)" va genera eroarea "**Too many variables**" (la evaluarea expresiei "1 L E S S x"). De asemenea, poate fi folosită și pentru a verifica dacă 2 întregi sînt în relația factorial între ele.

Această definiție se citește în modul următor:

Pentru a găsi un y astfel încît x factorial—al y pentru x dat
găsește un y între 1 și x
astfel încît y factorial x este adevărată.

Definirea proprietății de divizibilitate "**are-divizor**":

x a r e — d i v i z o r i f y î n t r e (z x) &
T I M E S (y z x)

relația poate fi folosită pentru a verifica dacă un întreg are divizori (nu este număr prim).

Pentru a găsi toate nmerele divizibile dintr-un interval:

&. a l l (x : x î n t r e (*întreg—1 întreg—2*) **&**
x a r e — d i v i z o r)

furnizează toți întregii divizibili între *întreg—1* și *întreg—2*.

Definirea celui mai mare divizor comun "**CMMDC**"

1) Cel mai mare divizor comun a 2 numere întregi egale
este valoarea lor

2) Cel mai mare divizor comun a 2 numere întregi pozitive, diferite
este cel mai mare divizor comun al întregului mai mic și diferenței lor

(x y) CMMDC z este adevărată dacă și numai dacă
x, y sînt întregi pozitivi și z este cel mai mare divizor comun

(x x) C M M D C x i f x I N T

(x y) C M M D C z i f x I N T & y I N T &
x L E S S y & S U M (x X y) &
(x X) C M M D C z

(x y) C M M D C z i f x I N T & y I N T &
y L E S S x & S U M (y X x) &
(X y) C M M D C z

condițiile "INT" verifică condițiile de utilizare ale relației definite.
Definirea parității "par"

x par este adevărată dacă x este divizibil cu 2

x par if TIMES (y 2 x) & y INT

Definirea relației "divizor—al"

x divizor—al y adevărată dacă x este întreg între 2 și y
și x divide y; x și y dați

x divizor—al y if x între (2 y) &
TIMES (x z y) & z INT

LISTE

1. Listele ca obiecte

În propoziții : *nume—obiect—1 nume—relație nume—obiect—2*, numele obiectelor pot fi liste de forma (*nume—1 nume—2* ...), atunci la întrebarea

&. w h i c h (x : x *nume—relație* *nume—obiect—2*) sau

&. w h i c h (x : *nume—obiect—1 nume—relație* x)

răspunsul va fi lista "(*nume—1 nume—2* ...)"
Listele pot fi considerate ca mulțimi ordonate și numărabile în care oricare element se poate repeta de un număr nedeterminat de ori.
Atenție : La întrebarea :

&. i s (*nume—obiect—1 nume—relație nume—n*) sau

&. i s (*nume—n nume—relație nume—obiect—2*)

dacă *nume—n* este membru al listei ca obiect, atunci răspunsul va fi "**NO**" dacă în baza de date nu există o propoziție de exact acest tip.

2. Accesul la membrii unei liste

A) Liste de lungime fixă
( ) este o listă vidă.
Pentru a obține un membru al unei liste se folosesc întrebările "**which**" (sau "**is**" pentru verificare) punînd variabile în locurile necesare.
Elementele unei liste pot fi liste sau liste de liste ș.a.m.d.
Argumentele unei relații sînt termeni unde un termen poate fi :

— o constantă (un nume) (limitat la maximum 60 caractere)
— un număr
— o variabilă

Observație : Comanda "**accept**" în lucrul cu liste
Propoziția : (*lista—1—nume—1* ...) *nume—relație* (*lista—2—nume—2* ...) este reprezentată în forma prefixată cerută de **accept**, adică :
nume—relație . ((*lista—1—nume—1* ...) (*lista—2——nume—2* ...))

B) Liste de lungime nedeterminată

O listă este în general de forma (x1 ⋮ x2 ... xn) unde
cap ⋮ coadă

primul element x1 se numește capul listei.
lista (x2 ... xn) se numește coada listei.

Coada unei liste cu un unic element este o listă vidă ( ).
Pentru procesarea listelor PROLOG are o primitivă specială notată cu simbolul "|" care se citește "urmat de".
Propoziția :

(x|y)

se citește "(x|y)" este o listă în care elementul x este urmat de o listă y
Propoziția :

(x y|z)

se citește "(x y|z) reprezintă o listă de 2 elemente urmate de o listă-rest z
Propoziția :

((x y)|z)

se citește "((x y)|z)" reprezintă o listă care începe cu o sublistă cu 2 elemente
Propoziția :

((x | y) | z)

se citește "((x | y) | z)" reprezintă o listă care începe cu o sublistă care are cel puțin un element (x)
Definirea relației "**aparține—lui**" folosind primitiva "|"

x aparține—lui y este adevărată dacă și numai dacă
x este un element al listei y

x aparține—lui (x | y)

x aparține—lui (y | z) if
x aparține—lui z

La întrebarea :

&. which (x : x aparține—lui (*nume—1 . . .*))

răspunsul este : toate elementele care sînt de tipul *nume—n*
Lista răspunsurilor se termină cînd z devine lista vidă ( )
Această definiție se citește astfel :

Pentru a găsi un element dintr-o listă nevidă (y | z)
  răspunsul este x = y
  sau găsește un element x în coada listei z
Pentru a verifica dacă un element x se află într-o listă nevidă (y | z)
  verifică dacă x = y
  sau verifică dacă x aparține cozii z a listei
Pentru a găsi o listă nevidă care conține elementul x
  răspunsul este lista (x | z)
  sau găsește o listă z care conține elementul x și răspunsul este (y | z) unde y este o variabilă care nu aparține listei z
  (numărul listelor care conțin elementul x este infinit)

3. Lungimea listelor

Definirea relației "**are—lungimea**"
x are—lungimea y adevărată dacă și numai dacă
y reprezintă numărul elementelor listei x

( ) a r e — l u n g i m e a 0

(x | y) a r e — l u n g i m e a z i f

y a r e — l u n g i m e a X &

S U M (X 1 z)

Această definiție se citește astfel:
Pentru a găsi lungimea unei liste date L
dacă L = ( ) lungimea este z = 0 sau
dacă L = (x | y) găsește lungimea X a listei y
și lungimea listei L este z = X + 1
Pentru a găsi o pereche (L, n) astfel încît L are—lungimea n
răspunsul este L = ( ) și n = 0 și
găsește o pereche (x, y) astfel încît x are—lungimea y
și răspunsul este L = (z | x) și n = y + 1
Pentru a găsi o listă L de lungime dată z
dacă z = 0 lista este L = ( ) sau
găsește o pereche (x, y) astfel încît x are—lungimea y
și n = y + 1 atunci lista este L = (z | x) unde z este o variabilă care nu aparține listei x

Definirea relației inverse "**lungimea—lui**"

0 l u n g i m e a — l u i ( )

x l u n g i m e a — l u i (y | z) i f

x I N T & 0 L E S S x &

S U M (X 1 x) & X l u n g i m e a — l u i z

Această definiție se citește astfel:
Pentru a găsi o listă L (de variabile) de lungime x dată
dacă x = 0 atunci L = ( ) sau
verifică că x este întreg și x > 0 și
X = x − 1 și găsește o listă z de lungime X
atunci L = (y | z) unde y este o variabilă care nu se află în lista z

Condiția "**0 L E S S x**" este logic redundantă dar este necesară pentru a evita o recursie infinită dacă x < 0.

Atenție: La folosirea relației "**are—lungimea**" primul argument trebuie dat, altfel se obține un număr infinit de răspunsuri.
La folosirea relației "**lungimea—lui**" primul argument trebuie dat altfel apare eroare "**Too many variables**" la evaluarea condiției **INT.**

Ambele relații "**are—lungimea**" și "**lungimea—lui**" pot fi folosite pentru verificare (ambele argumente date).

4. Unificarea

PROLOG rezolvă problemele printr-o metodă numită unificare.
Prin unificare 2 propoziţii devin identice prin atribuire de valori variabilelor, cu condiţia că numele aceleiaşi variabile trebuie înlocuit cu aceeaşi valoare în toate locurile în care apare, dar, dacă o variabilă poate fi lăsată fără a i se atribui o valoare, atunci ea rămîne aşa.

Exemplu : x EQ y este adevărată numai dacă cele 2 argumente sînt identice sau pot fi făcute identice prin unificare
''*număr* E Q *număr*'', ''*nume* E Q *nume*'' şi
''x EQ x'' sînt adevărate dar
''*număr—1* E Q *număr—2*'', ''*nume—1* E Q *nume—2*'' şi
''x E Q y'' sînt false dacă *număr*—1≠*număr*—2, *nume*—1≠≠.*nume*—2 şi variabila x este de tip (formă) diferit de variabila y (nu pot fi unificate).

5. Seturi de răspunsuri ca liste

În multe cazuri un set de propoziţii poate fi scris mai compact cu ajutorul listelor. Setul :

nume—obiect—1 *nume—relaţie* *nume—obiect*
. . . . .   (a)
nume—obiect—n *nume—relaţie* *nume—obiect*

poate fi scris în forma :

(*nume—obiect*—1 . . .*nume—obiect—n*) *nume—relaţie nume—obiect* (b)

La întrebarea : &. a l l (x : x *nume—relaţie* *nume—obiect*)
răspunsul este (a) *nume—obiect*—1
. . . . .
nume—obiect—n
No (more) answers
(b) (*nume—obiect*—1 . . . *nume—obiect—n*)
No (more) answers

În cazul setului de propoziţii (a) se poate obţine un răspuns de tip (b) prin utilizarea relaţiei ''**isall**'' cu întrebarea :

&. w h i c h (x : x i s a l l (y : y *nume—relaţie* *nume—obiect*))

Relaţia ''**isall**'' transformă setul de răspunsuri într-o listă.
Definirea relaţiei ''**maxim—a**''
x maxim—a y adevărată dacă x este cel mai mare număr din lista y.
trebuie considerate 2 cazuri :
y are un singur element
y are cel puţin 2 elemente

```
x m a x i m — a (x)
x m a x i m — a (y z | X) i f
    Y m a x i m — a (z | X) &
      m a x i m — d i n t r e (Y y)
```

Definirea relației "**ultim**"

x ultim y adevărată dacă x este ultimul element din lista y trebuie considerate 2 cazuri :

ultimul element al unei liste cu un element este elementul însuși

ultimul element al unei liste cu cel puțin 2 elemente este ultimul element al cozii listei

x u l t i m (x)

x u l t i m (y z | X) i f

x u l t i m (z | X)

Definirea relației "**vecini—pe**"

(x y) vecini—pe z adevărată cînd x și y sînt unul lîngă altul în lista z

trebuie considerate 2 cazuri :

x și y sînt primele 2 elemente ale listei z

x și y sînt vecini aparținînd cozii listei z

(x y) v e c i n i — p e (x y | z)

(x y) v e c i n i — p e (z | X) i f

(x y) v e c i n i — p e X

Definirea relației "**in**" utilizînd relația "**aparține—lui**"

x în y adevărată dacă x este un elemnt al listei y sau x este un element al unei subliste a listei y

x i n y i f x a p a r ț i n e — l u i y

x i n y i f z a p a r ț i n e — l u i y &

x i n z

Definirea relației "**in**"

x i n (x | y)

x i n (y | z) i f x i n z

x i n ((y | z) | X) i f x i n (y | z)

REGULI COMPLEXE

1. Negația

Uneori condițiile pe care trebuie să le satisfacă datele se exprimă mai natural prin exprimarea condițiilor pe care nu trebuie să le satisfacă.
Relația "**not**" este de forma:

not *C*

și este adevărată dacă *C* este falsă, unde *C* este o condiție simplă sau o conjuncție de condiții delimitată între paranteze.
Această propoziție se citește astfel:

Nu este adevărată propoziția *C* cu variabilele x1 ... xn

unde x1 ... xn sînt variabile ale *C* care nu apar în întrebare sau reguli (sînt variabile locale). Variabilele care apar și în *C* și în altă parte sînt variabile globale.

Restricții: O relație "**not**" poate fi folosită doar pentru verificare. În momentul evaluării propoziției "**not** *C*" variabilele globale trebuie să aibă valori. Deci într-o întrebare, "**not** *C*" trebuie să fie precedată de alte condiții pentru toate variabilele globale care sînt folosite, pentru atribuirea de valori variabilelor globale. Analog pentru folosirea "**not**" într-o regulă.

Exemplu: "**not** x **EQ** y" este adevărată dacă x este diferit de y sau nu pot fi unificate.

Definirea proprietății "**impar**" utilizînd "**par**":

x **impar** **if** x **INT** & **not** x **par**

Definirea proprietății "**prim**" utilizînd "**are—divizor**"

x **prim** **if** x **INT** &

not x **are—divizor**

2. Relația „isall"

O condiție "**isall**" este de forma:

L **isall** (*V* : *C*)

unde *V* este o listă de variabile, *C* o conjuncție de condiții și *L* este o variabilă sau o listă.
Această propoziție se citește astfel:

Pentru a rezolva condiția "*L* **isall** (*V* : *C*)"
construiește lista tuturor răspunsurilor la întrebarea (*V* : *C*) în ordinea inversă găsirii lor
apoi unifică *L* cu lista răspunsurilor

(PROLOG generează o listă în ordine inversă datorită eficienței)
implementării pe calculatoare)

L este lista tuturor răspunsurilor de tipul *V* pentru care condiția *C* este adevărată (pot exista variabile locale) în ordine inversă găsirii lor.

Restricții : La evaluarea condiției **isall** toate variabilele globale trebuie să aibă valori (la întrebări sau reguli)

Atenție : La evaluarea condițiilor "**isall**" și "**not**" PROLOG nu verifică dacă variabilele globale au căpătat valori. O utilizare incorectă a acestor condiții va genera răspunsuri greșite.

Utilizare generală : "**isall**" este utilizată pentru a compacta într-o listă toate răspunsurile la o întrebare de tipul "a l l (*V* : *C*)"

Utilizare pentru verificare : În general trebuie evitată utilizarea condiției "**isall**" pentru verificare, deci cu *L* listă dată, datorită ordinii elementelor din *L*

Dar condiția "**isall**" poate fi utilizată, în siguranță, la verificarea unei liste vide.

Definirea relației "**intersecția**"

x intersecția (y z) este adevărată dacă x este lista tuturor elementelor care apar în listele y și z

```
x intersecția (y z) if x isall
      (X : X aparține—lui y &
            X aparține—lui z)
```

poate fi folosită numai pentru găsirea intersecției a *2* liste (din cauza restricției pentru "**isall**"). Dacă y și z conțin fiecare un element comun care se repetă în fiecare listă, acesta va apărea repetat și în lista **x**.

Definirea relației "diferență"

x diferența (y z) este adevărată dacă x este lista tuturor elementelor care aparțin lui y și nu aparțin lui z

```
x diferența (y z) if
    x isall (X : X aparține—lui y &
                 not X aparține —lui z)
```

Definirea relației "**termen—in**"

```
x termen—in (y z) if
          x aparține—lui y
x termen—in (y z) if
          x aparține—lui z
```

Definirea relației "**uniunea**"

x uniunea (y z) este adevărată cînd x este lista tuturor elementelor care aparțin lui y sau z

```
x uniunea (y z) if x isall
    (X : X termen—in (y z))
```

Definirea relației "**sublistă—a**"

x sublistă—a y este adevărată dacă toate elementele lui x sînt elemente a lui y (x este intersecția dintre x și y)

x **s u b l i s t ă — a** y **i f**

z **i n t e r s e c ț i a** (x y) &

( ) **d i f e r e n ț a** (x z)

Definirea relației "**reuniunea**"

x reuniunea (y z) este adevărată dacă x este lista tuturor termenilor care apar în y sau z fără a se repeta dacă y și z nu conțin termeni repetați comuni

x **r e u n i u n e a** (y z) **i f**

X **u n i u n e a** (y z) &

Y **i n t e r s e c ț i a** (y z) &

x **d i f e r e n ț a** (X Y)

Primitiva PROLOG "**LST**" este de forma

x **L S T**

și este adevărată dacă x este o listă (x poate fi lista vidă sau o listă de forma (y | z) unde y și z sînt termeni de orice formă).

Definirea relației "**individual—in**"

x individual—in y este adevărată dacă x este un element care nu este de tip listă în y sau într-o sublistă a lui y

x **i n d i v i d u a l — i n** (x | y) **i f n o t** x **L S T**

x **i n d i v i d u a l — i n** ((y | z) | X) **i f**

x **i n d i v i d u a l — i n** (y | z)

x **i n d i v i d u a l — i n** (y | z) **i f**

x **i n d i v i d u a l — i n** z

Definirea relației "**conține**"

x conține y este adevărată dacă y este lista tuturor elementelor individuale din lista x

x **c o n ț i n e** y **i f** y **i s a l l**

(z : z **i n d i v i d u a l — i n** x)

(lista y va păstra ordinea din x dacă ultima regulă din definiția "**individual—in**" este pusă prima, adică se generează căutare de la coadă la cap).

3. Relația „forall"

O condiție "**forall**" este de forma :

(**f o r a l l** *C* **t h e n** *C'*)

unde *C* și *C'* sînt condiții simple sau conjuncții de condiții.

Parantezele indică unde se termină C' și începe condiția următoare.
Semnificația logică a condiției ''**forall**'' este:

pentru toate variabilele locale x1 ... xn pentru care C este adevărată, atunci este adevărată și C'

Restricții: Toate variabilele globale care apar în condiție ''**forall**'' trebuie să li se fi atribuit valori înainte de evaluare (în întrebări sau reguli).

Atenție: PROLOG nu verifică dacă variabilelor globale li s-au atribuit valori înainte de evaluare. O utilizare incorectă va genera răspunsuri greșite.

Condiția ''**forall**'' se citește astfel:

Pentru a verifica condiția (**forall** C **then** C')
  răspunde la întrebarea ''**all** (**x1** ... **xn** : C)
  și pentru fiecare set de valori ale variabilelor locale verifică dacă C' este adevărată
  dacă s-a găsit un set de valori astfel încît C' este falsă atunci abandonează căutarea și răspunde: condiția ''**forall**'' falsă
  dacă toate soluțiile pentru C sînt și soluții pentru C' atunci răspunde: condiția ''**forall**'' adevărată.

Condiția ''**forall**''

(**forall** C **then** C')

este echivalentă cu

not (C & **not** C')

deoarece aceasta este adevărată numai dacă nu există nici o soluție pentru:

C & **not** C'

și pentru această condiție nu există soluție dacă nu există soluția pentru C sau toate soluțiile pentru C sînt și soluții pentru C' adică nu sînt soluții pentru ''**not** C' ''

[$(C \Rightarrow C') \Leftrightarrow \neg(C \wedge \neg C')$ este tautologie]

Definirea relației ''**sublistă—a**'':

```
x sublistă—a y if
    (forall z aparține—lui x)
        then z aparține—lui y)
```

Definirea relației ''**egal**'':

```
x egal y if x sublistă—a y &
            y sublistă—a x
```

Definirea relației de ordonare:

x ordonată este adevărată dacă pentru toate perechile vecine (y, z) condiția y mai—mic—egal z este adevărată

```
x ordonată if
    (forall (y z) vecini—pe x
        then y mai—mic—egal z)
```

Definirea unei liste de numere pozitive:

```
x num—poz if
    (forall y aparține—lui x
        then 0 LESS y)
```

Definirea relației "**disjuncte**":

disjuncte (x y) este adevărată dacă x și y sînt liste fără elemente comune:

```
a) disjuncte (x y) if
       not (z aparține—lui x &
           z aparține—lui y)
b) disjuncte (x y) if
       () isall (z : z aparține—lui x &
                    z aparține—lui y)
c) disjuncte (x y) if
       (forall z aparține—lui x
           then not z aparține—lui y)
```

4. Relația „or"

O condiție "**or**" este de forma

(**either** *C* **or** *C'*)

unde *C* și *C'* sînt condiții simple sau conjuncții de condiții.
Parantezele indică unde se termină *C'* și începe condiția următoare.
Semnificația logică a condiției "**or**" este:

sau *C* sau *C'* [C v C']

Condiția "**or**" se citește astfel:

Pentru a rezolva o condiție "(**either** *C* **or** *C'*)"
rezolvă condiția *C*
sau rezolvă condiția *C'*

O relație definită prin 2 reguli poate fi definită printr-o singură regulă, utilizînd relația "**or**".

Definirea relației "**aparține—lui**":

```
x aparține—lui (y | z) if
    (either x EQ y
        or x aparține—lui z)
```

Definirea relației "**are—lungimea**":

```
x are—lungimea y if
    (either x EQ () & y EQ 0
        or x EQ (z | X) &
            X are—lungimea Y &
            SUM (1 Y y))
```

Definirea relației "**uniunea**" :

```
x  uniunea (y z)  if  x  isall (X :
        (either X  aparține-lui y
               or X  aparține-lui z))
```

Definirea relației "**ultim**" :

```
x  ultim  y  if
    (either y EQ (x)
         or y EQ (z | X) &
              x  ultim  X)
```

Definirea relației "**vecini—pe**"

```
(x y) vecini-pe z if
      (either z EQ (x y | X)
           or z EQ (Y Z | X) &
              (x y) vecini-pe (Z | X))
```

5. Expresii

Observație : Pentru a utiliza expresiile se încarcă programul „**EXPTRAN**" (6 blocuri) care recunoaște și compilează expresii.

Formal o expresie este :

o constantă	o expresie aritmetică
un număr	un apel de funcție
o variabilă	o listă de expresii

Expresiile pot fi folosite în relații "=" și în relații de expresii. O relație "=" este de forma :

E1 = *E2*

unde *E1* și *E2* sînt expresii.

Această relație este o extindere a relației **EQ**. Cînd "=" este rezolvată se evaluează întîi expresiile *E1* și *E2*, apoi ele sînt comparate prin **EQ**. Dacă una din expresii este o variabilă, atunci acesteia i se atribuie valoarea celeilalte expresii.

Expresiile aritmetice și apelurile de funcții sînt tipuri speciale de liste.

O expresie aritmetică este o listă de forma.

(*expresie operator expresie*)

unde operator poate fi :

*	pentru înmulțire
% sau /	pentru împărțire
+	pentru adunare
~ sau —	pentru scădere

deci expresiile aritmetice sînt liste de 3 elemente (parantezele sînt obligatorii) unde al 2-lea element este un operator.

Dacă expresiile din expresia aritmetică sînt la rîndul lor expresii aritmetice atunci pentru eliminarea confuziilor se introduc paranteze suplimentare (dacă acestea sînt necesare) conform regulilor :

— operatori * și % sînt evaluați cu prioritate față de + sau ~

— în cazul în care există operatori * și % învecinați expresia este evaluată în ordine de la stînga la dreapta

— în cazul în care există operatori + sau ~ învecinați expresia este evaluată în ordine de la stînga la dreapta

(conform regulilor aritmetice obișnuite)

Atenție : "%" și "—" sînt operatori principali ; "/" și "~" fiind sinonimi acceptați

"—" trebuie încadrat între separatori de cuvinte de exemplu blank-uri)

Un apel de funcție este o listă de forma :

(*R E1* ... *En* — 1) (1)

unde *E1* ... *En — 1* sînt expresii și *R* este o relație n-ară a cărui nume trebuie să fi fost declarat ca funcție cu comandă :

function *R*

Expresiile sînt primele n — 1 argumente ale unei relații de forma :

R (*V1* ... *Vn — 1* x) (2)

unde *V1* ... *Vn — 1* sînt valorile expresiilor *E1* ... *En — 1*

Apelul unei funcții (1) returnează valoarea variabilei x care s-ar obține dacă condiția (2) ar fi rezolvată.

Atenție : Dacă o relație nu este declarată ca funcție înainte de a fi folosită într-o expresie, atunci interpretorul de expresii nu va compila apelul funcției în expresia respectivă și o va lăsa sub formă de listă. Interpretorul de expresii atenționează asupra acestui fapt prin mesajul "*R* **assumed not to be a function**". Dacă apelul funcției a fost într-o întrebare, atunci se va obține un răspuns greșit. Dacă apelul funcției s-a făcut într-o propoziție atunci corectarea se face în modul următor :

— se declară *R* ca funcție : **function** *R*

— se editează propoziția fără modificări : se apelează editorul apoi se iese imediat din editor fără a efectua modificări (Editorul decompilează forma compilată a propoziției apoi o recompilează, dar ținînd seama de faptul că *R* a fost declarat ca funcție).

Pentru a găsi funcțiile care au fost declarate se folosește

which (x : x func) sau

list func

Dacă o relație declarată ca funcție este ștearsă atunci automat va fi ștearsă și propoziția care o declară ca funcție.

Expresiile aritmetice și apelurile de funcții pot fi îmbricate una în alta fără restricții.

Pentru ca PROLOG să compileze argumentele unei relații ca expresii se folosește "#" plasat între numele relației și lista argumentelor de tip expresie.

O propoziţie (condiţie) folosind expresii este de forma :

R # (*E1* ... *En*)

unde *R* este numele relaţiei şi *E1* ... *En* sînt expresii "#" semnalizează faptul că *E1* ... *En* nu sînt argumente normale ale relaţiei 2, ci unele sau toate conţin operatori aritmetici sau apeluri de funcţii. Expresiile pot fi folosite ca argumente ale unei relaţii numai în forma prefixată.

Relaţia de egalitate :

E1 = *E2*

este echivalentă cu

EQ # (*E1* *E2*)

Formele relaţionale între expresii :

R # (*E1* ... *En*)

sînt compilate sub forma unei condiţii complexe "#"

(*R* (*T1* ... *Tn*)) # (*conjuncţie—de—condiţii*)

Evaluarea conjuncţiei—de—condiţii va genera valori pentru variabilele din termenii *T1* ... *Tn* astfel încît aceştia devin valori ale expresiilor originale *E1* ... *En*.

Formele compliate ale expresiilor aritmetice folosesc relaţiile aritmetice definite astfel :

+ (x y z) **if SUM** (x y z)

— (x y z) **if SUM** (y z x)

* (x y z) **if TIMES** (x y z)

% (x y z) **if TIMES** (y z x)

Observaţii : formele compilate pot fi afişate dacă se utilizează propoziţia "**rel—form**"

&. add (rel — form)

atunci la listare sau editare relaţiile care folosesc expresii vor fi afişate în forma lor compilată.

"**rel—form**" nu opreşte compilarea expresiilor, ci decompilarea acestora la listare sau editare.

Pentru a anula efectul "**rel—form**" se foloseşte

&. kill rel — form

Cînd o formă relaţională între expresii este evaluată se calculează întîi conjuncţia—de—condiţii care calculează valorile expresiilor-argument.

Forma generală a întrebărilor-expresii este :

(*expresie*)

care are ca unic răspuns valoarea expresiei

Definirea funcţiei "factorial" :

```
1 factorial 1
x factorial y if
    1 LESS x &
    z = (x~ 1) &
    z factorial X &
  y = (X * x)
function factorial
```

Definirea funcţiei "lungime" (similară relaţiei "are—lungimea") :

```
() lungime 0
(x | y) lungime z if
      y lungime X &
      z = (X + 1)
function lungime
```

Definirea funcţiei "suma" ; suma numerelor dintr-o listă :

```
() suma 0
(x | y) suma z if
      y suma X & z = (X + x)
function suma
```

Definirea funcţiei "medie" ; media numerelor dintr-o listă :

```
x media y if
    y = ((suma x) / (lungime x))
function medie
```

6. Interogarea utilizatorului

"**is—told**" este o relaţie care generează întrebări către utilizator avînd ca singur argument o listă de variabile ; în mod similar listei de variabile din întrebările "**which**" cu diferenţa că în cazul acesta PROLOG este cel care întreabă utilizatorul şi primeşte un răspuns de la el.

Observaţie : pentru a utiliza relaţia "**is—told**" se încarcă fişierul „**TOLD**". (3 blocuri)

La evaluare, secvenţa de elemente din lista de variabile este afişată, urmată de "**?**" indicînd că PROLOG este în aşteptarea unui răspuns la condiţia pusă.

Tipul şi efectul răspunsului :

yes condiţia (propoziţia) se presupune a fi adevărată, la căutarea înapoi (back tracking) întrebarea nu va fi repetată ;

no condiția (propoziția) se presupune a fi falsă :

ans... "..." este o secvență de valori, cîte una pentru fiecare variabilă diferită din lista de variabile.
Condiția "**is-told**" este adevărată pentru setul de valori ale variabilelor dat ca răspuns. A n-a valoare din secvența—răspuns va fi atribuită celei de a n-a variabile din lista de variabile în ordine de la stînga la dreapta.
Prin backtracking, întrebarea poate fi repetată pentru căutarea unor soluții alternative pînă la furnizarea răspunsului "**no**" sau "**just ...**"

just... Analog "**ans...**" cu excepția că întrebarea nu va fi repetată la căutare backtracking. Se consideră că soluția dată "..." pentru condiția "**is—told**" este unică sau ultima.

Condiția "**is—told**" poate fi folosită în întrebări sau reguli precum și în depanarea sau dezvoltarea unor programe de aplicații.

7. Comentarea programelor

Pentru a introduce comentarii în baza de date (program) există 2 posibilități :

— se adaugă propoziții despre o relație "**comentariu**"
— se adaugă comentarii ignorate de compilator, folosind primitiva PROLOG "/ *"

Primitiva PROLOG "/ *" este de forma :

/ * (*comentariu*)

Orice propoziție care utilizează această primitivă este ignorată de PROLOG în timpul evaluării. O propoziție "/*" este întotdeauna adevărată.
Definirea funcției "**factorial**"

```
function factorial
   1 factorial 1 if
     /* (poate fi folosită doar pentru găsirea
        factorialului unui număr întreg dat)
   x factorial y if
     /* (variabile (x întreg)
                   (y valoare)) &
      1 LESS x & x INT
      y = (x * f actorial (x~ 1))
```

funcția "**factorial**" a putut fi apelată recursiv numai datorită faptului că a fost declarată anterior ca funcție.

Relația "**comentariu**" poate fi folosită în forma :

nume—relație **comentariu** (*comentariu*)

unde "(*comentariu*)" reprezintă comentariul la relația "*nume—relație*"

Pentru a găsi comentariu unei relații se folosește

which (**x** : *nume—relație* **comentariu x**)

Pentru a șterge toate comentariile se folosește

kill comentariu

Atenție : Comentariile ocupă memorie și consumă timp de calcul. Pentru ca programele să fie mai mici și să ruleze mai repede comentariile trebuie eliminate (păstrînd eventual o variantă a programului comentat).

Definirea funcției "**div**" :

div (x y z) este adevărată dacă z este partea întreagă a expresiei x/y (x, y întregi).

```
div (x y z) if x INT & y INT &
                    TIMES (y X x) & X INT z
function div
```

MANIPULAREA LISTELOR

1. Relația „concat"

Definirea relației "concat"

concat (x y z) este adevărată dacă z este rezultatul concatenării listei x la începutul listei y

concat (( ) x x)
concat ((y | z) x (y | X)) **if**
concat (z x X)

Atenție: z nu este pur și simplu lista (x | y)
Pentru a descompune o listă:

which (x y : **concat** (x y (*listă*)))

Pentru a exclude răspunsurile care conțin liste vide:

which ((x | y) (z | X) :
concat ((x | y) (z | X) (*listă*)))

Pentru a descompune o listă în locul în care primul element al listei se repetă:

which ((x | y) (x | z) :
concat ((x | y) (y | z) (*listă*)))

Dacă a 2-a listă începe cu un anumit element e:

which (x(e | y):
concat (x(e | y) (*listă*)))

Definirea relației "**ordonată**":

x ordonată este adevărată dacă x este o listă ordonată.

Există 4 cazuri:

x este o listă vidă
x este o listă cu un element
x este o listă cu cel puțin 2 elemente y și z identice
x este o listă cu cel puțin 2 elemente y și z astfel încît y LESS z

( ) **ordonată**
(x) **ordonată**
(y y | z) **ordonată if**
(y | z) **ordonată**
(y z| x) **ordonată if**
y LESS z &
(z | x) **ordonată**

Definirea relației "**extrage**" :

extrage (x y z) este adevărată dacă lista z este lista y din care a fost extras elementul x și toate duplicatele sale.

Există 3 cazuri :

y este o listă vidă

y este o listă al cărei prim element este x

y este o listă al cărei prim element este y diferit de x

```
extrage (x () ())
extrage (x (x|y) z) if
        extrage (x y z)
extrage (x (y|z) (y| X)) if
        not x EQ y &
        extrage (x z X)
```

Definirea relației "**compactată**" :

y compactată z este adevărată dacă z este lista conținînd toate elementele listei y fără repetare

Există 2 cazuri :

y este o listă vidă

y este o listă cu un prim element x atunci z trebuie să aibă de asemenea, ca prim element x dar coada listei z trebuie să fie versiunea compactată a cozii listei y adică să nu cuprindă nici un duplicat al elementului x.

```
() compactata ()
(x| y) compactată (x | z) if
        extrage (x y z)   &
        y compactată z
```

Această relație poate fi folosită pentru a extrage toate duplicatele dintr-o listă—răspuns la întrebarea "**isall**" de forma :

x isall (*R* : *C*) & x compactată y

Definirea relației "**față**" :

față (x y z) este adevărată dacă lista y este formată din primele x elemente ale listei z

```
față (x y z) if concat (y Xz) &
        y are—lungimea x
```

sau dacă lungimea x este dată :

```
față (x y z) if y are — lungimea x &
        concat (y Xz)
```

Definirea relației "**segment–in**" :

x segment–in z este adevărată dacă lista x este segmentul inițial al listei z

(x | y) **s e g m e n t – i n** z **i f**

c o n c a t ((x | y) X z)

Definirea relației "**segment–fin**" :

x segment–fin y este adevărată dacă lista x este segmentul final al listei y

(x | y) **s e g m e n t – f i n** z **i f**

c o n c a t (X (x| y) z)

Atenție : segmentele inițiale și finale trebuie să fie liste nevide.

Definirea relației "**segment–al**"

x segment–al y este adevărată dacă lista x este un segment al listei y

x **s e g m e n t – a l** y **i f**

z **s e g m e n t – f i n** y &

x **s e g m e n t – i n** z

Definirea relației "**inversă**" :

x inversă y este adevărată dacă y este lista nevidă x in ordine inversă

(x) **i n v e r s ă** (x)

(x y| z) **i n v e r s ă** X **i f**

(y | z) **i n v e r s ă** Y &

c o n c a t (Y (x) X)

Definirea relației "**șterge**" :

șterge (x y z) este adevărată dacă z este lista y cu elementul x extras (șters)

ș t e r g e (x y z) **i f**

c o n c a t (X (x | Y) y) &

c o n c a t (X Yz)

Definirea relației "**permutare–a**" :

x permutare–a y este adevărată dacă lista x este o permutare a listei y

( ) **p e r m u t a r e–a** ( )

(x | y) **p e r m u t a r e – a** z **i f**

ș t e r g e (x z X) &

y **p e r m u t a r e – a** X

poate fi folosită pentru generare sau verificare

Definirea relației **"sortată"**:

x sortată y este adevărată dacă y este o permutare ordonată a listei x

x sortată y if y permutare — a x &

y ordonată

poate fi folosită (dar ineficient) pentru ordonarea listei x dată.

Definirea relației **"ultim—al"**:

x ultim—al y este adevărată dacă x este ultimul element al listei y

x ultim — al y if concat (z (x) y)

Definirea relației **"aparține lui"**:

x aparține — lui y if

concat (z (x| X) y)

Definirea relației **"listă—putere"**:

x listă—putere y este adevărată dacă y este lista tuturor sublistelor ale listei x (lista vidă este și ea o listă care trebuie să apară o singură dată)

x listă — putere (( ) | y) if

y isall (z: z segment — al x)

Definirea relației **"simetrică"**:

x simetrică este adevărată dacă x este o listă care poate fi citită la fel de la cap la coadă și invers

x simetrică if x inversă x

Definiția relației **"vecini—pe"**:

(x y) vecini — pe z if

concat (X(x y| Y) z)

Definiția relației **"șterge"**:

șterge (x (x | y) y)

șterge (x (y | z) (y | X)) if

șterge (x z X)

Definiția relației **"separă"**:

separă (x y z X) este adevărată dacă z și X sînt rezultatul separării listei y în 2 componente, astfel încît z are lungimea x

1) **separă (x y z X) if**

concat (z X y) &

z are — lungimea x

```
2) separă (x y z X)  if
       x lungimea—lui z &
       concat ( z X y)
3) separă (0 x ( ) x)
   separă (x (y | z) (y | X) Y) if
       0 LESS x &
       SUM (Z 1 x) &
       separă ( Z z X Y)
```

2. Sortarea (ordonarea) listelor

Pentru a sorta (ordona) în mod eficient (timp de calcul minim) o listă se va folosi metoda "divide et impera". Această metodă constă în a diviza o problemă complexă într-o serie de probleme mai mici, rezolvate separat, ale căror soluţii sînt reunite pentru găsirea soluţiei problemei iniţiale.
O listă cu un element este sortată.
O listă mai mare este separată în 2 subliste cu ajutorul relaţiei "**separă**"
apoi cele 2 subliste sînt sortate şi
apoi cele 2 subliste sînt reunite într-o singură listă,
astfel încît lista finală să fie sortată. Aceasta se realizează cu relaţia unește.

Definirea relaţiei "**unește**" :
unește (x y z) este adevărată dacă lista z sortată este reuniunea listelor sortate x şi y
trebuie considerate cazurile :

- una din cele 2 liste este vidă
- dacă ambele liste sînt nevide atunci există cazurile :
  - primul element al fiecărei liste sînt egale
  - primul element al primei liste este mai mic decît primul element al celei de-a 2-a liste sau vice-versa.

Relaţia "**mai—mic**" poate fi definit utilizînd primitiva LESS sau relaţia "**is—told**".
Lista originală este despicată în 2 subliste de lungimea aproximativ egală cu ajutorul relaţiei "**despică**" definită prin relaţia "**separă**".
Programul complet este :

```
x sortare y if
  (either ( ) sortare ( )
          or (z) sortare (z))
(x y |z) sortareX if
     despică ((x y | z) Y Z) &
     Y sortare x1  & Z sortare y1 &
     unește (x1  y1  X)
```

```
unește (() x x)
unește (x () x)
unește ((x | y ) (x | z) (x x | X)) if
    unește (y z X)
unește ((x | y) (z | X) (x | Y)) if
    x mai—mic z &
    unește (y (z | X) Y)
unește ((x | y) (z | X) (z | Y)) if
    z mai—mic x &
    unește ((x | y) X Y)
despică (x y z) if
    separă ((div (lungime x) 2) x y z)
```

unde relația "separă" este definită în varianta 3).

Definirea relației "mai—mic" :

```
1) x mai—mic y if x LESS y sau
2) x mai—mic y if
       (x mai—mic y) is—told
```

Definiția relației "partiție" :

partiție (x y z X) este adevărată dacă fiecare element al listei x care este mai mic ca y este în lista z și toate celelalte elemente ale lui x sînt în lista X.

```
partiție (() x () ())
partiție ((x | y) z (x | X) Y) if
    x LESS z &
    partiție (y z X Y)
partiție ((x | y) z X (x | Y)) if
    not x LESS z &
    partiție (y z X Y)
```

Definirea relației "sortare—rapidă" folosind relația "partiție" (asemănătoare cu relația "sortare"):

```
() sortare—rapidă ()
(x) sortare—rapidă (x)
(x y | z) sortare—rapidă X if
    partiție ((y | z) x Y Z) &
    Y sortare—rapidă x1 &
    Z sortare—rapidă y1 &
    concat (x1 (x | y1) X)
```

Definirea relației "**uni—sort**" :

(x y) uni—sort z este adevărată dacă z este lista sortată a listei y de lungime x

```
( 0 ( ) ) uni—sort( )
( 1 (x) ) uni—sort (x)
( x y) uni—sort z if
        1 LESS x &
        uni—desp ( (x y) X Y) &
        X uni—sort x1 &
        Y uni—sort y1 &
        unește (x1 y1 z)
uni—desp ( (x y) (z X) (Y Z) ) if
        z = (d i v x 2 ) &
        Y = (x ~ z) &
        separă (z y X Z)
```

(la aceasta se adaugă regulile pentru "**unește**" și "**separă**").

3. Funcții și liste

Primitiva " | " nu se folosește niciodată într-o expresie urmată de o listă de tip apel de funcție. Pentru a specifica coada unei liste ca un apel de funcție se utilizează apelul de funcție "**CONS**" care adaugă un element în fața listei "**CONS**" este definită în "**SIMPLE**" și este recunoscută automat ca funcție în expresii fiind definită în modul :

```
CONS (x y (x | y))
```

Observație : "**SIMPLE**" mai conține relațiile :

"**APPEND**" echivalentă cu "**concat**"

"**ON**" echivalentă cu "**aparține—lui**"

ANALIZA GRAMATICALĂ

1. Analiza listelor de cuvinte (I)

O listă de cuvinte este propoziție dacă poate fi separată în 2 subliste, prima fiind de tip propoziție—substantiv și cea de a 2-a fiind de tip propoziție—verb. Deci structura gramaticală a unei propoziții **P** este o listă de tipul **(P x y)** unde x este o propoziție—substantiv și y este o propoziție—verb. Atunci relația "**propoziție**" (care arată dacă o listă de cuvinte x este propoziție) trebuie să fie de forma :

```
x propoziție (P y z) if
  APPEND (X Y x) &
  X prop—subst y
  Y prop—verb z
```

O propoziție substantiv este formată dintr-un articol urmat de o expresie—substantiv.

Programul pentru recunoașterea unei propoziții—substantiv **PS** este de forma :

```
x prop—subst (PS y z) if
  APPEND (X Y x) &
  X articol y &
  Y exp—subst z
```

Definiția pentru "**articol**" este :

```
  (x) articol (A x) if
      x dicționar ART
```

Relația "**dicționar**" reprezintă baza de date pentru cuvintele cunoscute adică este un vocabular care cuprinde lista de cuvinte și tipul asociat lor.

Atenție : Programul poate recunoaște numai cuvintele aflate în dicționar. O propoziție care cuprinde cuvinte care nu sînt în dicționar nu va fi analizată corect.

Secvența de program pentru relația "**dicționar**" și articole este de forma :

```
un dicționar ART
  o dicționar ART
niște dicționar ART     etc.
```

Programul pentru relația "exp—subst" este:

```
(x) exp—subst (S x) if
    x dicționar SUBST
(x) exp—subst (ES y z) if
    APPEND (X Y x) &
    X exp—subst y &
    Y adjectiv z .
```

Secvența de program pentru relația "**dicționar**" și substantive este de forma:

```
    subst—1 dicționar SUBST
    subst—2 dicționar SUBST   etc.
```

Programul pentru relația "**adjectiv**" este:

```
(x) adjectiv (ADJ x) if
    x dicționar ADJ
```

Secvența de program pentru relația "**dicționar**" și adjective este de forma:

```
    adjectiv—1 dicționar ADJ
    adjectiv—2 dicționar ADJ   etc.
```

Programul pentru recunoașterea unei propoziții-verb **PV** este de forma:

```
x prop—verb (PV y z) if
  APPEND (X Y x) &
  X exp—verb y &
  Y exp—subst z
```

Programul pentru relația "exp—verb" este:

```
(x) exp—verb (V x) if
    x dicționar VERB
(x) exp—verb (EV y z) if
    APPEND (X Y x) &
    X exp—verb y &
    Y adverb z
```

Secvența de program pentru relația "**dicționar**" și verbe este de forma:

```
    verb—1 dicționar VERB
    verb—2 dicționar VERB   etc.
```

Secvența de program pentru relația "**dicționar**" și adverbe este de forma:

```
    adverb—1 dicționar ADV
    adverb—2 dicționar ADV
```

Relațiile definite anterior sînt ineficiente datorită folosirii relației **APPEND.**

2. Analiza listelor de cuvinte (II)

O altă metodă de a analiza propozițiile este :

x p r o p o z i ț i e (P y z) i f

(x X) p r o p — s u b s t y &

X p r o p — v e r b z

Semnificația logică este :

x este o propoziție de tipul (P y z) dacă
diferența dintre x și coada sa X este
o propoziție—substantiv y și
X este o propoziție—verb z

Regula pentru relația "**prop—subst**" trebuie să unifice o pereche de liste cu structura gramaticală :

(x y) p r o p — s u b s t (P S z X) i f

(x Y) a r t i c o l z &

(Y y) e x p—s u b s t X

Semnificația logică este :

— diferența dintre x și coada sa y este o propoziție—substantiv de tipul (PS z X)

— dacă există Y astfel încît diferența dintre x și sublista coadă a sa (Y) este un articol z și diferența dintre Y și y este o expresie substantiv X

În acest mod folosirea relației "**APPEND**" a fost evitată.
Faptul că lista care este diferența dintre x și y este descompusă în liste care cuprind articolul și expresia—substantiv este arătat implicit în reprezentarea acestor subliste ca diferența dintre Y și y pentru un Y anume. Y trebuie să fie o sublistă din coada lui x mai mare decît y.
Regula pentru articol trebuie să recunoască diferența dintre o pereche de liste ca reprezentînd o listă conținînd un unic element : articolul.

((x | y) y) a r t i c o l (A x) i f

x d i c ț i o n a r A R T

Definiția relației "**exp—subst**" este :

((x | y) y) e x p — s u b s t (S x) i f

x d i c ț i o n a r S U B S T

(x y) e x p — s u b s t (E S z X) i f

(x Y) e x p — s u b s t z &

(Y y) a d j e c t i v X

Definiția relației "**adjectiv**" este :

((x | y) y) a d j e c t i v (A J y) i f

y d i c ț i o n a r A D J

Definirea relației "prop—verb" este :

```
(x y) prop — verb (PY z X) if
      (x Y) exp — verb z &
      (Y y) prop — subst X
```

Definiția relației "exp—verb" este :

```
(x y) exp — verb (EY z X) if
      (x Y) exp — verb z &
      (Y y) adverb X
```

Definiția relației "adverb" este :

```
((x | y) y) adverb y if
          y dicționar ADV
```

Setul de relații definite este utilizabil atît pentru analiza gramaticală a unor propoziții date cît și pentru generarea propozițiilor noi, folosind cuvinte din dicționar.
Pentru generare se poate folosi întrebarea :

```
which (x : număr lungimea — lui y &
          x propoziție y)
```

Setul de relații definit este utilizabil doar pentru propoziții cu structuri simple, dar poate fi extins.

3. Substituția relației „APPEND”

Tehnica de înlocuire a relației "APPEND" cu diferența dintre 2 liste poate fi folosită în orice definiție în care relația "APPEND" este folosită pentru a genera toate soluțiile rezultate din divizarea unei liste sau pentru a alipi 2 liste generate de alte condiții.
Definirea relației "d i f — i n v" :
"d i f — i n v" returnează o reprezentare generală a diferenței unei perechi de liste, în ordine inversă

```
(x) dif — inv ((x| y) y)
(x| y) dif — inv (z X) if
      y dif — inv (z (x | X))
```

Definiția relației "inversă" :

```
x inversă y if x dif — inv (y( ))
```

Definirea relației "sortare—rapidă" :

```
x sortare — rapidă y if
      x dif — sort — rap (y ( ))
( ) dif — sort — rap (x x)
(x | y) dif — sort — rap (z X) if
      partiție (y x Y Z) &
      Y dif — sort — rap (z (x | x1)) &
      Z dif — sort — rap (x1 X)
```

TEHNICI DE PROGRAMARE

1. Unicitatea soluţiilor

Relaţia "!" atenţionează PROLOG că o relaţie (condiţie) are o soluţie unică.
Forma de utilizare este:

R ! (*T1* ... *Tn*)

în formă prefixată cu "!" plasat între numele relaţiei *R* şi lista argumentelor sale (*T1* ... *Tn*).
La găsirea unor soluţii pentru *R* PROLOG întrerupe căutarea altor soluţii.
Relaţia "!" poate fi utilizată în reguli şi întrebări.

2. Primitiva PROLOG „/"

"/" este o primitivă PROLOG care permite controlul backtrackingului şi trebuie citită ca adevărată sau ignorată. În evaluarea unei întrebări dacă "/" este adevărată atenţionează PROLOG să caute înapoi căi alternative pentru rezolvarea condiţiilor impuse care sînt înaintea "/".
În întrebări primitiva "/" poate fi înlocuită cu relaţia de soluţie unică dar în reguli "/" este mai eficientă.
În definiţii primitiva "/" atenţionează PROLOG că regula care urmează după "/" este ultima care poate fi folosită pentru a găsi o soluţie la condiţiile impuse chiar dacă mai există şi alte reguli care nu au fost testate.
Exemplu:

```
P if C &/
P if (C) is-told
```

are semnificaţia logică:

P este adevărată dacă
C este adevărată (eventual pentru un set oarecare de variabile)
sau *C* este confirmată de utilizator

datorită "/" a 2-a regulă nu se aplică dacă C este adevărată (deci utilizatorul nu este interogat).
Definirea relaţiei "**uneşte**" cu ajutorul primitivei "/"

```
uneşte (() x x) if/
uneşte (x () x) if/
uneşte ((x | y) (x | z) (x x | X)) if
      / & uneşte (y z X)
uneşte ((x | y) (z | X) (x | Y)) if
      x mai-mic z & / &
      uneşte (y (z | X) Y)
uneşte ((x | y) (z X) (z | Y)) if
      z mai-mic x &
      uneşte ((x | y) X Y)
```

"/" atenționează PROLOG să folosească numai o singură regulă din cele 5 în care este definită relația "unește".

3. Stiva de întrebări

Cînd PROLOG încearcă să rezolve o întrebare atunci generează o secvență de întrebări suplimentare și folosește regulile pentru rezolvarea condițiilor din întrebare. PROLOG memorează calea urmată, construind o stivă de întrebări. De fiecare dată cînd aplicînd o regulă se generează o nouă întrebare, această nouă întrebare este pusă în vîrful stivei, la baza stivei fiind întrebarea inițială. Cînd a fost găsită o soluție pentru întrebarea curentă există 3 cazuri:

- dacă întrebarea curentă este întrebarea inițială simplă, atunci afișează soluția și începe căutarea soluției următoare dacă soluția nu a fost semnalizată ca unică sau dacă nu s-a cerut o singură soluție
- dacă întrebarea este derivată dintr-o condiție intermediară atunci se caută soluția pentru aceasta (care este soluție și pentru întrebările anterioare) și cu soluția găsită se merge la condiția următoare. Întrebarea nu este retrasă din stivă decît dacă PROLOG știe că nu mai există soluții (nu mai există reguli de încercat în cadrul definițiilor respective) sau i s-a transmis această informație de către utilizator
- dacă întrebarea derivată este ultima atunci soluția la aceasta este și soluție pentru întrebarea inițială.

Stiva de întrebări ocupă loc în memorie alături de programe și module, PROLOG fiind un mare consumator de memorie. Așadar, acolo unde restricțiile de memorie sînt dure, se impune o mare atenție în formularea regulilor și întrebărilor.

Dacă PROLOG nu mai are spațiu suficient în memorie, atunci întrebarea în curs de evaluare este abandonată și se afișează mesajul de eroare "**No space left**". În acest caz trebuie șterse definițiile și modulele nefolosite. Întrebarea din vîrful stivei este întotdeauna retrasă dacă condiția care a generat-o a fost definită ca avînd soluție unică deoarece PROLOG nu efectuează backtracking pe această întrebare, deci "/" reduce dimensiunea stivei.

4. Definiții recursive prin coadă

În general la aplicarea regulilor recursive apare o creștere rapidă a stivei de întrebări.

Definițiile recursive prin coadă sînt un tip de definiții recursive care cuprind o singură regulă recursivă care este ultima din lista de reguli pentru definiția respectivă și în această ultimă regulă condiția recursivă este plasată la sfîrșit. Acest tip de definiție împiedică creșterea stivei.
Deci, o definiție recursivă prin coadă cuprinde o succesiune de reguli nerecursive și o regulă recursivă, ultima, care este de forma :

R (*T1* ... *Tn*) if *C1* & ... *Ck* & *R* (*T'1* ... *T'n*)

unde doar ultima condiție se referă la relația definită R.

Condiţiile *C1* ... *Ck* care o preced trebuie să fie definite astfel încît atunci cînd PROLOG le rezolvă ştie că nu mai există soluţii pentru niciuna din condiţii.

Definiţia recursivă prin coadă a soluţiei "**uneşte**" :

```
uneşte (( ) x x)
uneşte (x ( ) x)
uneşte ((x | y) (z | X) (Y | Z)) if
    alege ! ((x | y) (z | X) Y x1 y1) &
    uneşte (x1 y1 Z)
alege ((x | y) (z | X) x y (z | X)) if
    x mai—mic z
alege ((x | y) (z| X) z (x | y) X) if
    not x mai—mic z
```

Relaţia auxiliară "alege" este folosită pentru a forma 2 liste x1 şi y1 astfel încît de pe una din ele a fost retrasă valoarea mai mică dintre x şi z.

PROLOG a fost atenţionat asupra unicităţii soluţiei pentru relaţia "**alege**" cu ajutorul relaţiei "**!**". Aceasta se poate face şi folosind primitiva "/", plasată după condiţia "alege" din a 3-a regulă din definiţia relaţiei "**uneşte**", sau, plasată după prima regulă din definiţia relaţiei "**alege**".

Pentru a transforma o definiţie care cuprinde mai multe reguli recursive într-o definiţie recursivă prin coadă, metodologia este următoarea :

a) se comasează toate regulile recursive într-o singură regulă generală, ceea ce de altfel în general clarifică definiţia ;

b) apoi prevăzînd cîte o unică soluţie pentru fiecare din condiţiile regulei generale, se foloseşte relaţia de condiţie unică "**!**", sau, se foloseşte primitiva "/" plasată înainte de partea recursivă (ultimă) a regulei.

Definiţia recursivă prin coadă a relaţiei "**maxim—a**" :

```
x maxim (( ) x)
x maxim ((y | z) X) if
        Y maxim—dintre (y X) & / &
        x maxim (z Y)
x maxim—a (y | z) if
    x maxim (z y)
```

x maxim (z y) este adevărată dacă x este cel mai mare dintre y şi maximum listei z

A 2-a regulă a acestei definiţii arată că, cel mai mare dintre X şi maximum listei (y | z), este cel mai mare dintre maximum listei z şi maximum dintre y şi X

"/" atenţionează PROLOG asupra unicităţii soluţiei din regula recursivă. Definiţia semi-recursivă prin coadă a relaţiei "**sortare—rapidă**" :

```
( ) sortare—rapidă (x x)
(x) sortare—rapidă ((x | y) y)
(x y | z) sortare—rapidă (X Y) if
      partiţie ((y | z) x Z x1) &
      Z sortare—rapidă (X (x | y1)) & / &
      x1 sortare—rapidă (y1 Y)
```

Această definiţie cuprinde 2 apeluri recursive, dar ultimul satisface condiţiile impuse definiţiilor recursive prin coadă (condiţia recursivă este ultima). Prin urmare, prin aplicarea acestei definiţii stiva va creşte, dar numai pe jumătate din cît ar fi crescut în cazul unei definiri ineficiente. Aplicarea celui de-al 2-lea apel nu determină creşterea stivei.
Definiţia recursivă prin coadă a relaţiei "**minim—a**" :

```
x minim—a (y | z) if
      x minim (z y)
x minim (( ) x)
x minim ((y | z) X ) if
      minim—dintre ! (Y y X) &
      x minim (z Y)
```

această relaţie este definită într-un stil analog relaţiei "**maxim—a**"
Definiţia recursivă prin coadă a relaţiei "**sortare**" :

```
( ) sortare ( )
x sortare (y | z) if
      y minim—a x &
      şterge (y x X) & / &
      X sortare z
```

Definiţia recursivă prin coadă a relaţiei "**factorial**" :

```
RC—factorial (x 1 x)
RC—factorial (x y z) if
      RC—factorial # ((x * y) (y ~ 1) z)
  x factorial y if
      RC—factorial (1 y x)
```

Definiţia recursivă prin coadă a relaţiei "**partiţie**" :

```
partiţie (( ) x ( ) ( ))
partiţie ((x | y) z X Y) if
    (either x LESS z &
            X EQ (x | Z) & Y EQ x1
        or  not x LESS z &
            X EQ Z & Y EQ (x | x1)) &
    / & partiţie (y z Z x1)
```

5. Module

Modulele sînt colecţii închise de definiţii complete, cărora li se asociază 2 liste speciale :

- lista de export, care cuprinde numele relaţiilor definite în modul şi care pot fi folosite în exterior (în reguli sau module) (numărul acestor relaţii fiind mai mic sau egal cu numărul total al relaţiilor definite în modulul respectiv)
- lista de import, care cuprinde numele relaţiilor definite în exteriorul modulului, dar sînt utilizate în interiorul modulului, la definirea relaţiilor acestuia. Această listă mai cuprinde : numele tuturor obiectelor utilizate în interiorul modulului şi în exterior în întrebări legate de relaţiile exportate.

Avantajul utilizării modulelor constă în faptul că definiţiile sale sînt protejate la modificări şi ştergere (din exterior)

Toate numele care nu sînt specificate în cele 2 liste asociate modulului respectiv sînt nume locale. Dacă un astfel de nume este utilizat în exteriorul modulului, (într-un alt modul sau în spaţiul de lucru) acesta va fi utilizat de PROLOG ca un nume diferit. Aceasta se realizează prin construirea a cîte unui dicţionar separat, pentru fiecare modul şi unul pentru spaţiul de lucru. (Modulele sînt analogul subrutinelor din limbajele de programare Pascal şi C). Dicţionarele păstrează toate constantele utilizate în program. Definiţiile relaţiilor neexportate sînt inaccesibile în exteriorul modulului.

Observaţie : Formarea modulelor

— se încarcă programul **MODULES** (5 blocuri) :
  load MODULES
— se adaugă propoziţia :
  Module (*nume—modul* (*lista—1*) (*lista—2*))
— relaţia "**Module**" are argumente :
  numele modulului : *nume—modul*
  lista de export : *lista—1*
  lista de import : *lista—2*
— se introduce comanda :
  wrap *nume*
  baza de date din spaţiul de lucru va fi împachetată într-un modul şi apoi spaţiul de lucru se eliberează

Atenţie : "*nume*" ; "*nume—modul*" şi numele relaţiilor definite în modul trebuie să fie diferite între ele.

Comanda "**wrap**" utilizează propoziţia "**Module** . . ." pentru a afla numele modulului şi listele sale de export şi import după care o şterge.

Pentru a şterge un modul :

kill *nume—modul*

Pentru a modifica un modul, acesta se despachetează cu comanda :

unwrap *nume—modul*

Pentru a şterge „**MODULES**" din memorie se foloseşte comanda :

kill modules—mod

Pentru a lista un modul se poate intra în acesta cu primitiva PROLOG "**OPMOD**" :

O P M O D *nume—modul*

atunci promt-ul se modifică din "&." în "*nume—modul.*"
apoi primitiva PROLOG "**L I S T**":

L I S T *nume—relație* sau

L I S T A L L

listează modulul

Pentru a ieși din modulul respectiv se folosește primitiva PROLOG "**C L M O D**" care acceptă orice argument.

Primitivele PROLOG sînt valabile în toate modulele și în spațiul de lucru. (sînt exportate implicit peste tot)

Primitiva PROLOG "**K I L L**" șterge unul sau mai multe module. (este analog "k i l l")

Primitiva PROLOG "**CMOD**" returnează, într-o variabilă neasignată, numele modulului curent ("&" pentru spațiul de lucru)

Primitiva PROLOG "**CRMOD**" creează un modul, are 3 argumente:

— numele modulului nou creat
— lista de export
— lista de import

METALOGICĂ

1. Metarelații

Meta-relațiile sînt relații ale căror argumente sînt nume de relații sau variabile, de tipul :

nume—relație—sau—variabilă nume—metarelație nume—relație—sau —variabilă

O astfel de meta-relație este "**true—of**". Dacă primul argument este o variabilă, atunci, acesteia trebuie să i se fi atribuit o valoare în momentul evaluării, această valoare fiind numele unei relații.
În caz contrar se afișează mesajul de eroare "**Too many variables**", împreună cu condiția, care are o variabilă, în locul numelui relației.
Dacă al 2-lea argument este o variabilă, atunci aceasta reprezintă lista generală de argumente ale relației respective. Dacă al 2-lea argument este o listă, atunci PROLOG va încerca unificarea acesteia, cu lista argumentelor relației respective ale fiecărei propoziții de definiție. Deci o variabilă ca prim argument este o meta-variabilă care înlocuiește numele unei relații și o variabilă ca argument secund este o meta variabilă care reprezintă lista argumentelor.
Întrebarea :

all (x y : x **dict** & x **true—of** y)

returnează lista tuturor relațiilor și listele lor de argumente.
Condiția :

x **true—of** (y z)

herifică dacă y și z verifică relația x
Condiția :

nume—relație **true—of** (*nume—obiect* | x)

verifică dacă "*nume—obiect*" este în relația "*nume—relație*" cu o listă de argumente necunoscute **x**.
Condiția :

nume—relație **true—of** x

returnează lista x a argumentelor, care satisfac relația "*nume—relație*".
Meta-relația "**true—of**" permite generalizarea unor relații, prin faptul că, numele unor relații poate fi dat doar la cerere, în momentul evaluării regulilor.

Definirea relației "**sortare**":

```
sortare (() () x)
sortare ((x) (x) y)
sortare ((x y | z) X Y) if
    despică ((x y | z) Z x1) &
    sortare (Z y1 Y) &
    sortare (x1 z1 Y) &/&
    unește (y1 z1 X Y)
unește (() x x y)
unește (x() x y)
unește ((x | y) (z | X) (Y | Z) x1) if
      alege ! ((x | y) (z | X) Y y1 z1 x1) &
      unește (y1 z1 Z x1)
alege ((x | y) (z | X) x y (z | X) Y) if
      Y true-of (x z)
alege ((x | y) (z | X) z (x | y) X Y)
      not Y true-of (x z)
```

unde argumentul Y din definiția relației "**alege**" este numele relației de comparare.
Pentru sortarea unei liste se folosește:

```
which (x : sortare (listă x LESS)
```

unde "*listă*" este lista care trebuie sortată, x este lista sortată (rezultatul) și **LESS** este relația de comparare.
Definirea relației "**transformă**":

transformă (x y z) este adevărată dacă z este rezultatul transformării listei y utilizînd relația x

```
transformă (x () y)
transformă (x (y) z)
transformă (x (y z | X) Y) if
    x true-of (y z Z) &
    transformă (x (Z | X) Y)
```

În momentul folosirii relația x trebuie să fie cunoscută.
Definirea relației "**ordonată**":

```
() ordonată x
(x) ordonată y
(x y | z) ordonată X if
    X true-of (x y) &
    (y | z) ordonată X
```

relația de comparare reprezintă al 2-lea argument al relației.

Definirea relației "**implică**":

implică (x y z) este adevărată cînd al n-lea element din lista y este în relația binară x cu al n-elea element al listei z

```
implică (x ( ) ( ) )
implică (x (y | z) (X | Y) ) if
    x true—of (y X) &
    implică (x z Y)
```

Definirea relației "**Implică**":

Implică (x y z) este adevărată dacă: 2 liste de o structură arbitrară y și z, sînt în relația dată x dacă elemente corespondente non-liste din y și z sînt în relația x

```
Implică (x ( ) ( )
Implică (x (y | z) (X | Y) ) if
    not y LST & x true—of (y X) &
    Implică (x z Y)
Implică (x (y | z) (X | Y) ) if
    y LST &
    Implică (x y X) &
    Implică (x z Y)
```

Atenție: primul argument de tip listă trebuie să fie dat altfel "y LST" generează eroare

2. Meta-programe care verifică condițiile de utilizare

"**CON**" este o primitivă PROLOG de forma:

```
x CON
```

și este adevărată dacă, în momentul evaluării ei, x este o variabilă, căreia i-a fost atribuită deja o constantă.

Definirea relației "**IMPLICA**":

IMPLICA (x y z) este adevărată dacă relația „Implică" (x y z)" poate fi folosită și o lansează în execuție în caz afirmativ

```
IMPLICA (x y z) if
    x CON & x defined &
    Implică (x y z)
```

"**VAR**" este o primitivă PROLOG de forma:

```
x VAR
```

și este adevărată dacă, în momentul evaluării ei, x este o variabilă, căreia nu i-a fost atribuită încă nici o valoare.
Definirea relației "lung" :

x **l u n g** y **i f**

x **V A R** & y **a r e — l u n g i m e a** x

x **l u n g** y **i f**

n o t x **V A R** & x **l u n g i m e a — l u i** y

3. Programe care controlează alte programe

În PROLOG toate comenzile pot fi folosite ca relații în reguli și întrebări.
"**FAIL**" este o primitivă PROLOG de forma :

F A I L

și care este întotdeauna falsă.
Semnificația logică este :
fals
Comenzile "**add**" și "**delete**", utilizate ca relații unare sau binare, pot adăuga sau șterge în respectiv din baza de date.
De asemenea, ele pot fi utilizate pentru contorizarea numărului de utilizări ale unei relații, în decursul evaluării unei întrebări.
Pentru aceasta se inițializează un contor folosind o propoziție care definește relația "**contor**" :

nume—R **c o n t o r 0**

unde "*nume—R*" numele regulii urmărite trebuie să fie diferit de "*nume—relație*" sau "*nume—obiect*".
Apoi se adaugă o condiție suplimentară la regula urmărită de forma :

... (*regulă*) ... &

nume—R **i n c — c o n t o r**

unde relația "**inc—contor**" determină incrementarea contorului cu o unitate la fiecare utilizare a regulii.
Definiția relației "**iuc—contor**" :

x **i n c — c o n t o r i f**

(x **c o n t o r** y) **d e l e t e** &

S U M (y 1 z) &

(x **c o n t o r** z) **a d d**

deci la fiecare utilizare reușită a regulii, valoarea veche a contorului este ștearsă și se adaugă noua valoare a contorului, care este cea veche la care se adaugă 1.
În momentul evaluării condiției "**delete**" x are valoarea "*nume—R*", astfel încît va fi ștearsă propoziția :

nume—R **c o n t o r** y

şi lui y i se atribuie valoarea veche a contorului prin unificare, apoi este ştearsă propoziţia respectivă.
O utilizare a comenzii "**delete**" în forma:

d e l e t e (*nume—obiect nume—relaţie* x)

va şterge din baza de date prima propoziţie de acest tip.
Definirea relaţiei "**Sumă**" (analog relaţiei "**sumă**"):

```
x Sumă y if
    (total 0) add &
    (forall z ON x
        then z total—actual) &
    (total y) delete
x total—actual if
    (total y) delete &
    SUM (y x z) &
    ( total z) add
```

Avantajul acestei tehnici de folosire a bazei de date este că baza de date poate fi folosită, de exemplu, pentru a aduna o secvenţă de numere, date ca propoziţii despre o relaţie unară, fără a fi necesară formarea listei numerelor, în modul următor:

```
x SUMA y if
        ( total 0) add &
    (forall x true—of (z)
        then z total—actual) &
    (total y) delete
```

în baza de date existînd o secvenţă de tipul:

număr—1 nume—relaţie

număr—2 nume—relaţie e.t.c.

Observaţie: manipularea bazei de date cu ajutorul relaţiilor "**add**" şi "**delete**" este o operaţie lentă.
Atenţie la poziţionarea condiţiei "**add**" în reguli; "**add**" trebuie să fie după condiţiile care generează valori pentru variabilele care apar în argumentul ei.
Exemple:
1) evaluarea condiţiei:
x **EQ** *nume—obiect* & (x *nume—relaţie*) **a d d**
adaugă propoziţia "*nume—obiect nume—relaţie*"
2) evaluarea condiţiei:
(x *nume—relaţie*) **a d d** & x **EQ** *nume—obiect*
adaugă regula "x *nume—relaţie*"

3) evaluarea condiţiei :

x E Q *nume—relaţie* &

(*nume—obiect—1* x *nume—obiect—2*) a d d

adaugă propoziţia "*nume—obiect—1 nume—relaţie nume—obiect—2*"

4) evaluarea condiţiei :

(*nume—obiect—1* x *nume—obiect—2*) a d d &

x E Q *nume—relaţie*

va genera o eroare de sintaxă.

Atenţie : Teoretic o condiţie "**delete**" utilizată într-o regulă poate şterge însuşi regula la utilizarea ei. Pentru a evita această situaţie "**delete**" se utilizează într-o definiţie care poate şterge numai alte definiţii.

4. Relaţii unare folosite ca comenzi

Orice relaţie unară poate fi folosită ca o comandă, şi poate fi redenumită Exemple :

x t o t   i f x a l l

x e s t e   i f x i s

după adăugarea acestor reguli, "**tot**" şi "**este**" pot fi folosite în locul "**all**" şi "**is**". Redenumirea numelui unei relaţii printr-o regulă nu împiedică folosirea ei.

Pentru a adăuga la baza de date un răspuns la o întrebare "**tot**" se foloseşte :

t o t (*propoziţie—de—adăugat* : *conjuncţie—de—condiţii* &

(*propoziţie—de—adăugat*) a d d)

Definirea comenzii-relaţie "**Tot**" :

"**Tot**" preia o întrebare de tipul :

t o t (*propoziţie—de—adăugat* : *conjuncţie—de—condiţii*)

şi concatenează condiţii suplimentare la sfîrşitul listei de condiţii din "**all**"

(x : | y) T o t i f

c o n c a t ((x : | y) (& x a d d) z) &

z t o t

Relaţia "**Tot**" cere ca listă de variabile **x** să fie un termen unic, care este totodată o listă-model pentru propoziţia de adăugat bazei de date, pentru fiecare soluţie a întrebării găsite.

Primitiva PROLOG "**R**" este de forma :

R (x)

şi este adevărată dacă şi numai dacă **x** este un termen.

Această propoziție se citește astfel:

pentru a rezolva condiția de forma R (x)

verifică că x este o variabilă, căreia nu i s-a atribuit încă nici o valoare

citește un termen T de la consolă care este soluția unică a condiției (atribuie termenul T variabilei x)

Deci "**R**" se folosește pentru a citi termeni de la tastatură. ("**R**" afișează "**.**" și așteaptă introducerea datelor). Termenul poate fi un număr, o constantă, o listă sau o variabilă. Orice variabilă citită este imediat redenumită după reguli interne (vechea denumire nu este reținută).

Observație : În cazul în care la consolă se introduce o serie de termeni, separați de spații, atunci condiția "R" va prelua primul termen, o utilizare ulterioară a condiției "R" îl va prelua pe cel de al 2-lea ș.a.m.d. pînă la epuizarea listei. Apoi la următoarea evaluare a condiției "R" se va aștepta din nou introducerea unui termen de la consolă.

Exemplu : Argumentele comenzii "**add**" sînt citite utilizînd primitiva "**R**".

Primitiva PROLOG "**P**" este de forma :

P (*T1* ... *Tn*)

este adevărată dacă *T1* ... *Tn* sînt termeni.

Această definiție se citește astfel:

Pentru a rezolva condiția P (T1 ... Tn)

verifică că T1 ... Tn sînt termeni și dacă sînt

afișează T1 ... Tn la consolă

Poziția cursorului rămîne la sfîrșitul textului afișat.

Primitiva PROLOG "*PP*" este similară cu "*P*" cu deosebirea că după afișarea termenilor cursorul este poziționat la începutul liniei următoare

Exemplu : Toate mesajele de eroare sînt definite utilizînd primitivele "*P*" și "*PP*".

Definirea relației "**spune—este**" (versiune simplificată a relației "**is—told**") :

```
x spune—este if
    P true—of x &
    P(?) & R (y) &
    y EQ da
```

Definirea relației "**IMPLICA**" :

```
IMPLICA (x y z) if
    x CON &
    x defined &/&
    Implică (x y z)
IMPLICA (x y z) if
    PP (Implică nu poate fi folosită deoarece
        numele relației nu este cunoscut) &
    FAIL
```

Primitiva "/" împiedică folosirea celei de a 2-a reguli atunci cînd condițiile de aplicare a primei reguli au fost satisfăcute.

Primitiva "**FAIL**" împiedică satisfacerea condițiilor celei de a 2-a reguli atunci cînd aceasta ajunge să fie aplicată deoarece prin modul în care a fost scrisă condiția "**PP**" este adevărată (lipsa ei ar duce la concluzia că regula a 2-a este adevărată, ceea ce evident este fals).

Definirea relației "**spune—este**" (versiune a relației "**is—told**"):

```
x spune—este if
    P true—of x &
    P (?) & R (y) &
    x a—răspuns y
x a—răspuns răspuns if
    x val—var
x a—răspuns răspuns if
    x spune—este
(x | y) val—var if
    x VAR & / & R(x) &
    y val—var
(x | y) val—var if
    y val—var
```

Această relație afișează o întrebare, care este o listă de variabile și constante, și la care se răspunde cu "**răspuns**", în același mod, în care la întrebări "**is—told**", se răspunde cu "**ans**".

Prin backtracking, întrebarea va fi repetată, pînă cînd, PROLOG va căpăta alt răspuns decît "**răspuns**" (definiția nu conține răspunsuri de tip "**yes**" și "**just**").

Relația "**val—var**" parcurge lista primită de la consolă și atribuie fiecărei variabile din întrebare o valoare din lista răspuns în ordine. Primitiva "**VAR**" din prima regulă este folosită pentru selectarea variabilelor, cărora nu li s-a dat încă o valoare, iar regula a 2-a este folosită pentru a sări peste termenii din întrebare, care nu sînt variabile.

Definiția relației "**acceptă**" (similară comenzii "**accept**"):

```
x acceptă if
    P (propoziție pentru x) &
    x R &
    x răspuns
sfîrșit răspuns
x răspuns if
    x LST &
    x add & x acceptă
```

Această relaţie poate fi folosită pentru a introduce o serie de propoziţii oarecare cu privire la o anumită relaţie pînă la introducerea "**sfîrşit**".
Definirea relaţiei "**editare**" :

```
x editare if
    x list &
    P (Numărul propoziţiei?) &
    y R &
    x edit y
```

relaţie unară al cărei argument este *nume—relaţie*, pe care o listează după care cere numărul de ordine al propoziţiei sau regulii de editat.

5. Supervizorul

La orice moment de timp utilizatorul se află în interiorul unei relaţii definite circular (din care nu se poate ieşi).
Această relaţie se numeşte "*supervizor*" şi reprezintă o interfaţă între PROLOG şi utilizator. Supervizorul afişează "&" care atenţionează utilizatorul că se află în spaţiul de lucru şi "." care arată că este în aşteptarea unei relaţii unare urmată de argumentul său unic, apoi aplică relaţia argumentului respectiv, după care reia bucla infinită.
La citirea consolei, se încarcă un tampon de intrare, pe măsură ce utilizatorul introduce textul. După ce utilizatorul a semnalizat terminarea textului (cu "ENTER"), controlul revine supervizorului pentru analiza textului. Controlul revine imediat utilizatorului pentru a continua introducerea textului în 3 situaţii :
— dacă nu s-a tipărit nimic : se afişează numai "."
— dacă există ghilimele neînchise : analog anterior şi ceea ce se introduce în continuare se consideră ca făcînd parte din textul introdus anterior
— dacă există paranteze deschise care nu au fost închise. În acest caz, supervizorul tipăreşte "*număr*" în locul "&", număr reprezentînd numărul parantezelor lăsate deschise. Interpretarea textului se face numai după ce au fost închise toate parantezele.

Definiţia relaţiei "*supervizor*" într-o versiune simplificată este :

```
supervizor if
    P (&) & R (x) & R (y) &
    (either x true—of y
        or P (?)) &/&
    supervizor
```

(relaţia "**supervizor**" este recursivă prin coadă datorită primitivei "/" astfel încît nu încarcă stiva).
Pentru a utiliza relaţii cu mai multe argumente ca nişte comenzi se adaugă cîte o regulă suplimentară, pentru fiecare argument, care citeşte argumentele suplimentare cu ajutorul primitivei **R**.

Aplicații :
1) comanda "**edit**" :

```
x edit if y R & x edit y
x edit y if ...
```

prima regulă confirmă condiția "**true—of**" din supervizor, cînd edit este utilizat ca o comandă și determină citirea celui de-al 2-lea argument, aplicîndu-se apoi regula a 2-a.
2) comanda "**add**" :

```
x add if x NUM &/&
    y R & x add y
x add if x LST &/&
    32767 add x
x add y if ...
```

primitiva "/" asigură folosirea unei singure reguli din cele 2 enunțate după cum x este un număr sau o listă :
— dacă este un număr, atunci acesta reprezintă poziția la care *lista—propoziție* (citită înainte de aplicarea formei binare a relației) trebuie inserată
— dacă este o listă atunci aceasta este o propoziție care se adaugă în poziția 32767, adică de fapt la sfîrșit.
Definiția relației "**spune—este**" :

```
x spune—este if
      P true—of x &
      P (?) & R (y) &
      x este y
  x este da
  x este nu if FAIL
  x este doar if x val—var
```

această relație permite folosirea "**da**", "**nu**" și "**doar**" în același mod în care "**is—told**" folosește respectiv "**yes**", "**no**" și "**just**".
Definirea relației "**editare**" :

```
x editare if
    x list & P (Numărul propoziției ?) &
    y R &
    y răspuns x
nu răspuns x
x răspuns y if
    x INT & y edit x & y editare
```

relaţia listează relaţia după fiecare editare după care reia ciclul pînă cînd se răspunde "nu".

Definiţia relaţiei "**List**" :

```
s f î r ş i t  L i s t
x  L i s t  i f
     x  l i s t  &  y  R  &
     y  l i s t
```

relaţia listează definiţii de relaţii date în secvenţă încadrate între "**List**" şi "**sfîrşit**".

Definirea relaţiei **SUPERVIZOR** (extinderea relaţiei "**supervizor**") :

```
S U P E R V I Z O R  i f
   P (&)  &  R  (x)  &
   (e i t h e r  x  c o m a n d ă  y  &
              y  a r e — a r g u m e n t e  x  &
              (e i t h e r  x  t r u e — o f  z
                          o r  PP  (?) )
       o r  P P (C o m a n d ă  i n c o r e c t ă)
   & / &  S U P E R V I Z O R
0  a r e — a r g u m e n t e  ( )
x  a r e — a r g u m e n t e  (y | z)  i f
       y  R  &
       a r e — a r g u m e n t e  #  ( (x ~ 1)  z)
```

"**SUPERVIZOR**" foloseşte o relaţie ajutătoare "**comandă**" care stabileşte care relaţii pot fi folosite drept comenzi şi cîte argumente au acestea. Relaţia, "**comandă**" este definită ca o secvenţă de propoziţii de tipul :

nume—relaţie—1 c o m a n d ă *număr—argumente—1*

nume—relaţie—2 c o m a n d ă *număr—argumente—2* e.t.c.

SINTAXA PROLOG STANDARD

1. Atomi și clauze

Relațiile PROLOG pot returna numai valori logice deci acestea sînt predicate logice.

Atomii sînt liste de forma următoare:

— primul element al listei = capul reprezintă numele unui predicat
— următoarele elemente ale listei = coada reprezintă lista argumentelor predicatului respectiv

deci atomii sînt de forma:

(*nume—predicat argument—1 ... argument—n*)

unde "*nume—predicat*" este o constantă.
Unele predicate particulare nu au nici un argument
(în PROLOG acestea sînt **NEW**, **FAIL**, **ABORT**, **CLMOD**).
Astfel:

1) relația infixată: *argument—1 nume—relație argument—2*
devine atomul: (*nume—relație argument—1 argument—2*)
2) relația postfixată: *argument nume—relație*
devine atomul (*nume—relație argument*)
3) relația prefixată nume—relație argument—1 ... argument—n
devine atomul (*nume—relație argument—1 ... argument—n*)
4) relația fără argumente: *nume—relație*
devine atomul (*nume—relație*)

Atomul se citește astfel:

argument—1 și ... *argment—n* sînt în relația *nume—predicat* între ele.
Predicatele logice se definesc prin clauze
Clauzele sînt liste de atomi de următoarea formă:

— primul element al listei = capul reprezintă atomul de definit
— următoarele elemente ale listei = coada reprezintă definiția primului atom

deci clauzele sînt de forma:

((*atom—de—definit*)
(*a t o m — 1*)
....
(*a t o m — n*))

[(atom—1) ∧ ... ∧ (atom—n) ⇒ (atom—de—definit)]

Semnificația logică a unei clauze este :

atom—de—definit este adevărat, dacă
atom—1 este adevărat și
. . . .
atom—n este adevărat

Pentru definirea unui predicat pot fi necesare mai multe clauze, în general într-o anumită ordine, ca alternative .
Astfel :

1) propoziția : *nume—obiect—1 nume—relație nume—obiect—2*
devine clauza : ((*nume—obiect—1 nume—relație nume—obiect—2*))
care conține un singur atom : cel de definit și deci care este întotdeauna adevărată

2) regula : x **termen—in** (x | y)

devine clauza : ((**termen—in** x (x | y)))

3) regula : **concat** (( ) x x)

devine clauza : ((**concat** ( ) x x))

4) regula : x *nume—relație—1* y **if**

x *nume—relație—2* y & y *nume—relație—2* x

[(∀) x (∀) y ((x nume—relație—2 y) ∧ (y nume—relație—2 x)
⇒ (x nume—relație—1))]
devine clauza : ((*nume—relație—1* x y)
(*nume—relație—2* x y)
(*nume—relație—2* y x))

5) regula : x aparține—lui (y | z) if
x aparține—lui z
devine clauza : ((aparține—lui x (y | z))
(aparține—lui x z))

6) regula : **concat** ((x | y) z (X | Y)) **if**

concat (y z Y)

devine clauza : ((**concat** (x | y) z (X | Y))
(**concat** y z Y))

Dacă un predicat este definit prin mai multe clauze, atunci ordinea lor în baza de date este aceea în care au fost introduse.

În baza de date nu contează ordinea clauzelor referitoare la predicate diferite.

Exemple de condiții complexe :

1) condiția : **not** x *nume—relație*
devine atomul : (**NOT** *nume—relație* x)

2) condiția : not (x *nume—relație—1* y & y *nume—relație—2*)
devine atomul : (**NOT ?** ((*nume—relație—1* x y)
(*nume—relație—2* y)))

3) condiţia x **i s a l l** (y z : y *nume—relaţie—1* z &
z *nume—relaţie—2*)
devine atomul : (**I S A L L** x (y z) ((*nume—relaţie—1* y z)
(*nume—relaţie—2* z)))

4) condiţia : (**f o r a l l** *nume—obiect nume—relaţie—1* x &
x *nume—relaţie—2*
t h e n x *nume—relaţie*—3)
devine atomul (**F O R A L L** ((*nume—obiect nume—relaţie*—1 x)
(*nume—relaţie—2* X))
((*nume—relaţie*—3 x)))

5) condiţia : (**e i t h e r** x *nume—relaţie—1*
o r n o t x *nume—relaţie—2*)
devine atomul (**O R** ((*nume—relaţie*—1 x))
((**N O T** *nume—relaţie—2* x)))
(OR are 2 argumente care sînt liste de atomi)

6) condiţia : *nume—relaţie* **!** (x *nume—obiect*)
devine atomul (**!** *nume—relaţie* x *nume—obiect*)
unde "**NOT**", "**?**", "**!**", "**ISALL**', "**FORALL**", "**OR**" sînt primitive PROLOG. ("**?**" are drept unic argument o listă de atomi şi returnează adevărat dacă toţi atomii din listă sînt adevăraţi)

2. Alcătuirea bazelor de date

Introducerea unei clauze se poate face în 2 moduri :

& . (*clauză*)

& . **A D D C L** (*clauză*)

unde "**ADDCL**" este o primitivă PROLOG care poate fi folosită şi sub forma unui atom de tipul :

(**A D D C L** *clauză număr*)

unde număr este poziţia unei clauze după care va fi inserată noua clauză adăugată.

Definirea relaţiei "**Add**" (similară cu "**add**") :

```
( (A d d  X)
    ( I N T  X)
    ( R  Y)
    ( A D  D C L  Y  X) )
( (A d d  X )
    ( L S T  X)
    ( A D D C L  X  3 2 7 6 7) )
( (A d d  X  Y )
    ( S U M  Z  1  Y)
    ( A D D C L  X  Z) )
```

Ștergerea unei clauze se poate face utilizînd primitiva PROLOG "**DELCL**" :

& . **D E L C L** (*clauză*)

"DELCL" poate fi folosită și ca atom de forma :

(**D E L C L** *nume—relație număr*)

care șterge clauza du numărul de ordine "*număr*" din definiția "*nume—relație*".

Definirea relației "**Delete**" (similară cu "**delete**") :

```
((D e l e t e  X)
    (I N T  X)
    (R  Y )
    (D E L C L  Y  X))
((D e l e t e  X)
    (L S T  X)
    (D E L C L  X))
```

Primitiva PROLOG specială "**DICT**" este definită în fiecare modul (dicționar) și în spațiul de lucrú printr-o singură clauză cu un singur atom de forma :

((**D I C T** *simbol* (*listă—1*) (*Listă—2*) *nume*—1 ... *nume—n*))

unde simbol în spațiul de lucru este "*&*"
într-un modul este "*nume—modul*"

lista—1 în spațiul de lucru este lista vidă
într-un modul este lista—export
(spațiul—de lucru nu exportă nimic)

lista—2 în spațiul de lucru este lista tuturor termenilor exportați de modulele rezidente în memorie
într-un modul este lista—import
(spațiul de lucru importă toate relațiile exportate de toate celelalte module)

nume—1 ... *nume—n* reprezintă lista tuturor predicatelor care au fost utilizate (chiar dacă nu există clauze pentru ele) ; spațiul de lucru conține implicit primitiva "**?ERROR?**" de mascare a erorilor

Definiția relației "**r e z e r v a t**" (similar relației "**reserved**") :

```
((r e z e r v a t  (d i c t  f u n c  d a t a — r e l | X))
    (D I C T  Y  Z  X | x))
```

Primitiva PROLOG "**?**" este o relație unară al cărei argument este o listă de atomi.

Dacă la comandă : *&* . **?** (*atom—1 ... atom—n*)
se răspunde *&* .
atunci lista de atomi este adevărată (pentru un set oarecare de variabile ; toți atomii sînt adevărați)

Dacă la comanda : *&* . ? (*atom—1* ... *atom—n*)

se răspunde : ?

&.

atunci lista de atomi este falsă (există cel puțin un atom fals)

Primitiva PROLOG "?" este utilizată pentru interogarea bazelor de date.

Atenție : Faptul că PROLOG a răspuns că un predicat este fals nu înseamnă neapărat că el este într-adevăr fals. Acest răspuns trebuie de fapt interpretat astfel : "predicatul este nedeterminat" sau „baza de date este insuficientă pentru a arăta că predicatul este adevărat". Acest fapt este unul din limitările importante ale inteligenței artificiale. PROLOG încearcă pe orice cale permisă să valideze atomii (să găsească soluții pentru care sînt adevărați) dar construirea unei baze de date suficient de extinse cade în sarcina utilizatorului precum tot acestuia îi revine decizia finală privind falsitatea predicatelor în cazul bazelor de date insuficiente.

O utilizare alternativă a primitivei "?" este de forma :

& . ? (*atom—1* ... *atom—n* (**PP** *listă—variabile*))

care afișează soluțiile corespunzătoare listei de variabile.

Definirea relației "**este**" (similară cu relația "**is**") :

```
((este X) (? X)/(PP da))
((este X) (PP nu))
```

Definirea relației "**care**" (similară cu "**which**") :

```
((care (X | Y))
    (FORALL Y ((PP X)))
      (PP Nu (mai) sînt soluții))
```

unde X capul listei este o listă de variabile

și Y coada listei este o listă de atomi pe care trebuie să le satisfacă variabilele din X

3. Meta-variabile

O meta-variabilă trebuie să fie asignată în momentul evaluării părții de clauză pe care o înlocuiește. Valoarea atribuită meta-variabilei trebuie să fie corectă sintactic pentru partea de clauză reprezentată de ea.

A) Meta-variabile care înlocuiesc nume de predicate.

O meta-variabilă poate înlocui numele unui predicat dintr-un atom din coada listei-clauză (atom—condiție), dar niciodată numele predicatului care se definește.

Definirea relației "**implică**" :

```
((implică X ( ) ( )))
((implică X (Y Z) (x | y))
    (X Y x)
    (implică X Z y))
```

Definirea relației "**transformă**" :

((**t r a n s f o r m ă** X (Y) Y))

((**t r a n s f o r m ă** X (Y Z | x) y)

(X Y Z z)

(**t r a n s f o r m ă** X (z | x) y))

Definirea relației "**pereche**" :

(pereche x y z) este adevărată dacă x și y sînt liste de tipul (*listă*—1 — element—1 ... listă—1 — element—n) respectiv (listă—2 — element—1 ... listă—2—element—n) iar Z este ((listă—1 — element—1 ... listă—2—element—1) ... (listă—1—element—n listă—2—element—n))

((**p e r e c h e** ( ) ( ) ( )))

((**p e r e c h e** (X | Y) (Z | x) ((X Z) | y))

(**p e r e c h e** Y x y))

Definiția relației "**prod—scal**" :

(p r o d — s c a l X Y Z) este adevărată dacă Z este produsul scalar al listelor de numere X și Y, adică dacă X = = (număr—1—1 ... număr—1—n) și Y = (număr—2—1 ... număr—2—n) atunci Z = număr—1—1* număr—2—1 + ... + număr—1—n* număr—2—n)

((**p r o d — s c a l** X Y Z)

(**p e r e c h e** X Y x)

(**i m p l i c ă** **p r o d — p e r** x y)

(**t r a n s f o r m ă** **S U M** y Z))

((**p r o d — p e r** (X Y) Z)

(**T I M E S** X Y Z))

B. Metavariabile care înlocuiesc atomi
O metavariabilă poate înlocui un atom dintr-o listă—clauză, dar niciodată primul atom (de definit).

Definirea relației "**not**" :

((**n o t** X) X / **F A I L**)

((**n o t** X))

unde **X** trebuie să fie un atom unic.

La folosire prima regulă verifică atomul **X**. Dacă **X** este adevărat atunci "/" previne utilizarea clauzei a 2-a și **FAIL** determină răspunsul "**not X**" este fals. În caz contrar (**X** fals), se aplică regula a 2-a care arată (întotdeauna) că "**not X**" este adevărat.

C. Meta-variabile care înlocuiesc liste de atomi

O meta-variabilă poate înlocui o secvenţă de atomi dintr-o listă—clauză, eventual toată coada listei—clauză, dar niciodată primul atom (de definit).
Definirea relaţiei "**not**" :

```
((not X)(? X)/FAIL)
((not X))
```

această definiţie este asemănătoare celei anterioare cu diferenţa că **X** este o listă de atomi. Primitiva "**?**" verifică dacă conjuncţia de condiţii impusă de **X** este adevărată.
Definirea relaţiei "**OR**" :

```
((OR X Y) | X)
((OR X Y) | Y)
```

în această definiţie **X** şi **Y** sînt toată coada listei—clauză.
Definirea relaţiei "**IF**" :

```
((IF X Y Z) X/ | Y)
((IF X Y Z) | Z)
```

Semnificaţia logică este următoarea :

dacă **X** este adevărată atunci **Y** este adevărat
altfel (x este fals) **Z** este adevărat

Definirea relaţiei "**care**" (similară relaţiei "**which**") :

```
((care (X | Y))
   (? Y) ( P P X) FAIL)
((care (X | Y) )
   ( P P Nu (mai) sînt soluţii))
```

la utilizare se foloseşte întîi prima regulă şi "**(PP X)**" afişează prima soluţie la condiţia "**(? Y)**" după care "**FAIL**" determină prin backtracking căutarea soluţiilor alternative la "**(? Y)**" pînă la epuizarea acestora după care se aplică regula a 2-a care afişează mesajul de sfîrşit.
Definirea relaţiei "**una**" (similară relaţiei "**one**") :

```
((una (X | Y))
   (? Y)
   ( P P X "încă (d/n) ?")
   (R Z)
   ( IF (EQ Z "n")
      ( )
      ( FAIL)))
((una(X | Y))
   ( P P Nu(mai) sînt soluţii))
```

D. Meta-variabile care înlocuiesc liste de argumente.

Dacă un atom este înlocuit cu structura "(X | Y)" atunci Y este o metavariabilă care reprezintă lista argumentelor pentru atomul respectiv. În general această metodă se folosește cînd numărul argumentelor este necunoscut.

Definiția primitivei multi argument "Not" :

```
((Not | X)
      (? X) / FAIL)
((Not |  X))
```

Definirea primitivei "!" :

```
((! | X) / X)
```

Definirea primitivei "FORALL" :

```
((FORALL X Y)
      (NOT ?((? X) (NOT ? Y))))
```

Această definiție se citește astfel :

"FORALL X Y" este adevărată dacă "((? X) (NOT ? Y))" este falsă (nu are soluții) și este falsă dacă X nu are soluții sau orice soluții pentru X sînt și pentru Y.

4. Alte primitive PROLOG

Numărul total al primitivelor PROLOG variază cu implementarea pe diferite calculatoare. Unele din ele pot fi definite în funcție de alte primitive de nivel mai coborît în general inaccesibile utilizatorului. Definițiile lor pot fi listate cu "**LIST**" *nume—primitivă*".

Dintre primitivele nemenționate deja cele mai importante sînt :

"**CON:**" fișier deschis permanent pentru scriere și citire la consolă

"? ERROR ?" permite mascarea erorilor de către utilizator. Înainte de lansarea unui mesaj de eroare, supervizorul verifică dacă există clauze de definiție pentru acest predicat. Primul atom din clauze trebuie să fie de forma : (" ? **ERROR** ?" *număr atom*)
unde "*număr*" este codul erorii și "*atom*" este atomul unde a apărut eroarea respectivă.
Evaluarea atomului este înlocuită cu evaluarea clauzei " ? **ERROR** ?" corespunzătoare codului de eroare generat. Atenție : Execuția programului nu se oprește. Pentru a opri execuția se folosește primitiva "**ABORT**".
Exemplu :

```
(("? ERROR ?" număr X)
      (PP Apărut eroarea număr)
      (PP în atomul X)
      ABORT)
```

Numai una din clauzele "? **ERROR** ?" va fi folosită așadar dacă există mai multe trebuie folosită primitiva "/"

CREATE
OPEN
CLOSE

sînt utilizate pentru lucrul cu fișiere pe suport magnetic
sînt de forma :
(*nume—primitivă nume—fișier*)
unde *nume—fișier* este o constantă
CREATE — creează un fișier pentru scris
OPEN — deschide un fișier pentru citit
CLOSE — închide un fișier deschis
atomii respectivi sînt întotdeauna advărați, cu excepția **CLOSE** care poate fi fals dacă nu există fișier deschis.
Restricții : Nu pot fi deschise 2 fișiere simultan.

LISTP — utilizată pentru listarea bazelor de date în fișiere specificate.
este de forma :
(**LISTP** *nume—fișier listă—predicate*)
unde "*nume—fișier*" este numele unui fiș:er deschis pentru scris (cu **CREATE** sau implicit de exemplu : "**CON** :" și *lista—predicate* este lista predicatelor cerută la listare.
Atomul este întotdeauna adevărat cu excepția cazului în care în lista predicatelor există o primitivă PROLOG. care nu poate fi listată

CL relație care acceptă 1 sau 3 argumente.
În general se utilizează ca relație unară de forma :
(**CL** ((*nume—relație* | X) Y))
Pentru a afla răspunsurile pentru relația "*nume—relație*"
Atomul este fals dacă nu există clauze pentru "*nume—relație*". În loc de *nume—relație* se poate folosi o meta-variabilă.

5. Concluzii

PROLOG este un demonstrator automat de teoreme construit pe baza regulilor calculului logic propozițional, putînd trage toate concluziile logice ce decurg dintr-o bază de date tratată de PROLOG ca fiind adevărată.
Demonstrațiile automate sînt limitate de :

A) Baza de date : limitată — O propoziție adevărată în realitate, dar inexistentă în baza de date poate fi declarată de PROLOG ca fiind falsă.

B) Baza de date falsă — Propoziții false în realitate dar declarate ca adevărate în baza de date (premise false) pot genera rezultate false în realitate dar declarate ca adevărate de PROLOG precum și propoziții adevărate în realitate dar declarate ca false de PROLOG. Conform tabelei de adevăr a relației de implicație "falsul implică orice" : A fals (0)⇒B adevărat (1) sau A fals (0)⇒B fals (0)

Deci adevărul sau falsitatea unei prouoziții trebuie privită doar în contextul bazei de date respective.
Folosind tautologiile calculului propozițional se pot construi relații și meta-relații cu caracter general.

DERIVAREA FORMALĂ A FUNCȚIILOR

Se va defini o relație unară "D" astfel încît "D(X)" afișează derivata funcției X de o singură variabilă. Variabila de derivare nu poate fi X, Y, Z sau variabile care încep cu X, Y, Z deoarece acestea sînt variabile PROLOG, atunci se va nota aceasta cu t. Funcția X va fi procesată ca o listă în care toate eventualele ambiguități vor fi eliminate prin utilizarea parantezelor.

Definirea relației "D"

```
( (D  X)
    (V  X)
    (D R V X  Y)
    (P P D e r i v a t a)
    (P P  X)
    (P P  e s t e)
    (P P  Y) )
( (D  X)
    (P P  E r o a r e  s i n t a c t i c ă) )
```

unde "V" verifică ca X să fie o listă care nu conține variabile PROLOG și "DRV" calculează efectiv derivata X care este Y.
Dacă X conține variabile PROLOG atunci "DRV" nu poate fi evaluată și se aplică regula a 2-a. Analog dacă X nu conține variabile PROLOG dar "DRV" nu poate fi evaluată.

Definirea relației "V"

```
( (V  X)
  (L S T  X)
  (F O R A L L  ( (I N  X  Y) )
                ( (N O T  V A R  Y) ) ) )
```

Definirea relației "IN"

```
( (I N  X  Y)
  (O N  X  Y) )
( (I N  X  Y)
  (O N  X  Z)
  (I N  Z  Y))
```

Definirea relației "ON"

```
((O N (X | Y) X))
((O N (X | Y) Z)
  (O N Y Z))
```

La evaluarea relației "IN" X este o listă cunoscută și se generează Y toate elementele listei X verificate apoi în "V" pentru a nu fi variabile PROLOG.

Definirea relației "DRV"

A) dacă X este o expresie complexă, listă, încadrată în paranteze, inițiala sau parte a expresiei — lista inițiale atunci se elimină parantezele, se derivează, apoi se pun parantezele ; se elimină parantezele redundante.

```
((D R V (X) (Y))
  ( L S T X)
  ( D R V X Y))
```

B) dacă X este o constantă, nu depinde de t (variabila de derivare) atunci derivata constantei este 0

```
((D R V X (0))
  (N O T I N X t))
```

0 este încadrat de paranteze deoarece X poate fi o sublistă a listei inițiale.

C) dacă X este o sumă sau diferență de 2 subliste atunci derivata X este sumă sau respectiv diferența derivatelor sublistelor

```
((D R V X Y)
  ( C Z x y X)
  ( S x )
  ( D R V Z z)
  ( D R V y X1 )
  ( S M z x X1 Y))
```

relația "C" separă lista X în listele Z și y între care există semnul x ;
relația "S" verifică dacă x este + sau — și dacă da atunci derivata lui Z este z și
derivata lui y este X1 și
relația "SM" construiește derivata lui X care ste Y.

1) Definirea relației "C"

```
((C X Y Z x)
  ( C O N C A T X (Y | Z) x))
((C O N C A T ( ) X X))
((C O N C A T ( X | Y) Z (X | x))
  ( C O N C A T Y Z x))
```

2) Definirea relației "S"

((S +))

((S "–"))

deci relația "C" fragmentează lista pînă cînd relația "S" găsește semnul + sau –

3) Definirea relației "SM"

a) dacă una din liste este lista vidă

((SM X Y ( ) X))

b) dacă una din liste este (0)

((SM X + Y Y)

(N – E Q X (0))

Definirea relației "N–EQ"

"N–EQ" transformă un număr sau lista cu un singur număr, eliminînd semnele sau parantezele redundante

— dacă X este număr

((N – E Q X(X))

(N U M X))

— dacă X este o listă cu un singur număr

((N – E Q (X) (X))

(N U M X))

— eliminarea parantezelor redundante

((N – E Q ((X)) (X))

(N U M X))

— eliminarea semnului "+": în mod normal în "(*număr*)" *număr* se consideră pozitiv dacă nu există semnul "–"

((N – E Q (+ X) (X))

(N U M X))

— alipirea semnului "–" la număr

((N – E Q (" – " X) Y)

(S U M X Y 0))

— alipirea semnului "–" și eliminarea parantezelor redundante

((N – E Q (("–" X)) Y)

(S U M X Y 0))

c) dacă se adună sau scade "0"

((S M X Y Z x)

(N – E Q X (0))

(S M ( ) Y Z x))

d) dacă se adună 2 elemente identice : $X + X = 2 * X$

((S M X + X Y)
    (T M S '(2) * X Y))

e) diferența a 2 elemente identice este nulă

((S M X "−" X (0)))

f) dacă unul din elemente se poate da factor comun

$z * Y \pm Y = z * (Y \pm 1)$

((S M X Y Z x)
    (O P L S T Z y)
    (E X (z * y))
    (C L L S T z X1)
    (S M X1 Y (1) Y1)
    (T M S Y1 Z x))

Definirea relației "OPLST"

"OPLST" transformă o listă într-un șir prin eliminarea parantezelor

((O P L S T (X) X))

((O P L S T X X))

Definirea relației "E"

"E" definește comutativitatea produsului

((E (X * Y) (X * Y)))

((E (X * Y) (Y * X)))

Definirea relației "CLLST"

"CLLST" realizează operațiunea inversă relației "OPLST" ; transformă un șir într-o listă prin încadrarea în paranteze

((C L L S T X X)
    (L S T X))

((C L L S T X (X)))

g) dacă se poate da factor comun (analog f)

$(y * z) \pm (y * X1) = y * (z \pm X1)$

((S M X Y Z x)
    (E X (y * z))
    (E Z (y * X1))
    (C L L S T z Y1)
    (C L L S T X1 Z1)
    (S M Y1 Y Z1 x1)
    (C L L S T y y1)
    (T M S y1 x1 x))

h) coeficienţii numerici se pun uzual la început, deci se extrag din Z, cu sau fără semn, şi se introduc în capul listei X ; ulterior se realizează şi sumarea coeficienţilor dacă este posibil.

```
((SM X Y Z x)
  (E—N Z y z)
  (V—S z)
  ((OR ((EQ Y "—") (SUM y X1 0))
       ((EQ y X)))
  (AD X1 X Y1)
  (SM Y1 + z x))
```

relaţia "E—N" extrage y din Z fără sau cu semn, relaţia "V—S" verifică : restul listei Z este o listă sumă (eventual vidă) care începe cu un semn. — relaţia "OR" îl transformă pe y în — y dacă este cazul, relaţia AD îl adună pe y transformat în X1 în faţa X, apoi se adună restul.

Definirea relaţiei "E—N"

```
((E—N X Y Z)
  (OR ((EQ (x | z) X)(N—EQ x (Y)))
      ((EQ (x y | Z) X)
       (N—EQ(x y) (Y)))))
```

unde primul caz din "OR" număr fără semn,
al doilea caz din "OR" număr cu semn (2 simboluri)

Definirea relaţiei "V—S"

```
((V—S (X | Y))
   (S X))
((V—S ()))
```

Definirea relaţiei "AD"

— dacă se adaugă o listă vidă lista-rezultat nu se modifică

```
((AD X () Y)
  (CLLST X Y))
```

— dacă se adaugă 0 lista—rezultat nu se modifică

```
((AD X Y Z)
  (N—EQ Y (0))
  (CLLST X Z))
((AD X Y Z)
  (N—EQ X (0))
  (CLLST X Z))
```

— dacă se adaugă un număr fără semn

((A D X Y Z)
(E — N Y x y)
(Y — S y)
(S U M X x z)
(A D z y Z))

-- dacă se adaugă un număr într-o listă care începe cu un semn

((A D X (Y | Z) (X Y | Z))
(S Y))

— în general

((A D X Y (X + | Y)))

i) eliminarea semnului + redundant

X + (— x) = X — x = Y
sau X — (+ x) = X — x = Y
sau X + x = Y

((S M X Y (Z | x) y)
(S Z)
(O R ((C M — S Y Z) (S M X "—" x y))
((S M X + x y))))

Definirea relației "CM—S"
"CM—S" caută dacă Y Z sînt o pereche de semne + "—"

((C M — S + "—"))
((C M — S "—" +))

j) în general

((S M X Y Z x)
(C X Y Z x))

relația "SM" realizează parțial o simplificare a listelor sumă

4) Definirea relației "TMS"

a) produsul cu o listă vidă nu modifică rezultatul

((T M S X ( ) X))
((T M S ( ) X X))

b) produsul cu 1 nu modifică rezultatul

((T M S X Y X)
(N — E Q Y (1))
((T M S X Y Y)
(N — E Q X (1))

c) produsul cu 0 generează rezultat nul

```
( (T M S  X  Y  (0) )
   ( O R  (  (N – E Q  X  (0) ) )
           (  (N – E Q  Y  (0) ) ) ) )
```

d) produsul cu Y = – 1 schimbă semnele în lista sumă Y pentru a obţine rezultatul

```
( (T M S  X  Y  Z)
  (N – E Q  X  (– 1) )
  (E Q  Y  (x  | y) )
  (N O T  E Q  x / )
  (S – S  Y  ( z | X1) )
  (O R  ( (  E Q  z + )  (E Q  X1  Z) )
        ( (  E Q  ("–" * z | X1)  Z) ) ) )
```

relaţia "N–EQ" verifică X = – 1
relaţia "EQ" extrage primul semn = x din Y
relaţia "NOT" verifică ca Y să nu fie fracţie
relaţia "S–S" schimbă semnele în Y şi extrage primul semn = Z din ea.
relaţia "OR" elimină semnul redundant + dacă z = +

Definirea relaţiei "S–S"

```
( (S – S  X  Y)
  ( I S A L L   Z  x  (O N  X  y)
          (O R  ( (C* M – S   y  x) / )
                (  (  E Q  y  x) ) ) ) )
  ( I N Y L  Z  Y)   )
```

relaţia "ISALL" construieşte Z care este lista cerută Y dar în ordine inversă
relaţia "INVL" inversează (reface) ordinea în listă

Definirea relaţiei "INVL"

```
( ( I N V L  X   Y)
  (I S A L L  Y  Z  (O N  X  Z) ) )
```

e) produsul cu Y = – 1 schimbă semnele în lista–sumă X pentru a obţine rezultatul (analog d)

```
( (T M  S  X  Y  Z)
  ( N – E Q  Y  (– 1)
  ( T M S  Y  X  Z) )
```

se utilizează regula definită anterior

f) produsul a 2 elemente egale reprezintă elementul ridicat la puterea a 2-a

```
((TMS X X Y)
  (PT X (2) Y))
```

g) raportul a 2 elemente egale este 1

```
((TM S X (/Y) (1))
  (CLL S T Y X))
((T MS (1/ X) Y(1))
  (CLLST X Y ))
```

h) dacă unul din elemente se poate da factor comun

$X^y * X = X^{(y+1)}$ sau $Y^x * Y = Y^{(X+1)}$

```
((TMS X Y Z)
  (I-PT X Y Z))
((TMS X Y Z )
  (I-PT Y X Z))
```

operația de incrementare a exponentului o face relația "I—PT"
Definirea relației "I—PT"

```
((I-PT (X↑Y) Z x)
  (CLLST X Z)
  (CLLST Y y)
  (SM (1) + y z)
  (PT Z z x))
```

prima relație "CLLST" verifică existența factorului comun
relația "SM" incrementează exponentul
relația "PT" reface lista—expresie rezultat

i) coeficienții numerici se pun uzual la început deci se extrag din **Y**, fără sau cu semn și se introduc în capul listei **X**; ulterior se realizează și produsul coeficienților dacă este posibil

```
((TMS X Y Z)
  (E-N Y x y)
  (V-P y)
  (I M x X z)
  (T MS z y Z))
```

relația "E—N" extrage numărul **x** fără sau cu semn.
relația "V—P" verifica: restul listei **Y** este **y** o lista—produs, adică nu este listă—sumă, nu conține semnele + sau —

relaţia "IM" realizează x * X = z

relaţia "TMS" realizează rezultatul z * y = Z

Definirea relaţiei "V–P"

```
((V – P ( X | Y))
   ( O P X )
   ( F ORALL ((O N Y Z))
                 ((NOT S Z))))
```

relaţia "OP" verifica : lista (X | Y) începe cu o operaţie * sau / (înmulţire sau împărţire)

Definirea relaţiei "OP"

```
(( O P *))
(( O P /))
```

Definirea relaţiei "IM"

```
(( IM X Y Z)
   ( E – N Y x y )
   ( V – P y )
   ( TIMES X x z )
   ( TMS (z) y Z) )
(( IM X Y Z)
    (TMS (X) Y Z))
```

j) extragerea semnului "*" redundand

```
(( TMS X (* | Y) Z)
   ( TMS X Y Z))
```

k) extragerea semnului "+" redundand

```
(( TMS X ( + | Y ) Z )
   ( TMS X Y Z ) )
```

1) dacă X sau Y sînt liste—sumă operaţiile + sau – sînt prioritare astfel încît X respectiv Y trebuie încadrate în paranteze

```
(( T M S X Y Z)
   ( NOT V – P ( * | X ))
   ( TMS ( X ) Y Z ) )
(( T M S XY Z )
    (NOT V – P ( * | Y ))
    (TMS X ( Y ) Z ))
```

m) împărțirea unei fracții

X / Y / Z = X / (Y * Z) sau

(1 / X) * Y / Z = Y / (X * Z)

```
( ( T M S  X  Y  Z )
   ( I P  X  Y  Z ) )
( ( T M S  X  Y  Z )
    (I P  Y  X  Z  ) )
```

împărțirea o efectuează relația "**IP**"

Definirea relației "**IP**"

```
( (I P  ( X / Y )  ( /  | Z)  x )
   ( C L L S T  Y  y)
   ( T M S  y  Z  z)
   ( O R ( (E Q  z  (Y1 ) ) )
         ( (E Q  z  Y1 ) ) )
   ( T M S  X1 ( / Y1 )  x ) )
```

— relația CLLST extrage și formează lista Y = y

— relația "TMS" realizează produsul numitorilor Y * Z = z

— a 2-a folosire a relației "TMS" formează rezultatul

n) în general; împărțire

```
( ( T M S  X( /  | Y)  Z)
   (C  X / Y  Z ) )
```

o) în general; înmulțire

```
( ( T M S  X  Y  Z )
        ( C  X  *  Y  Z) )
```

relația "**TMS**" realizează parțial o simplificare a listelor—produs.

5) Definirea relației "**PT**"

a) X la puterea 0 este egal cu 1

```
( ( P T  X  Y  ( 1 ) )
   ( N — E Q  Y  (0) ) )
```

b) **X** la puterea 1 este egal cu **X**

```
( ( P T  X  Y  X )
   ( N — E  Q  Y  ( 1 ) ) )
```

c) eliminarea parantezelor redundante pentru exponent număr pozitiv

```
( ( P T ( X )  Y  ( X ↑ Z ) )
   ( N — E  Q  Y  (Z) )
   ( L E S S  0  Z ) )
```

d) ridicarea la putere a unui număr cu exponent

```
(X ↑ Y) ↑ Z = X ↑ (Y * Z)
((P T(X ↑ Y) Z (X ↑ x))
    (CLLST Y y)
    (TMS y Z z)
    (OR( (EQ y (x))(LESS 0 x))
         ((EQ y x ))))
```

e) în general termenii trebuie încadrați în paranteze

```
((PT (X) Y Z)
  (PT (X) (Y) Z))
((PT X (Y) Z)
  (PT (X) (Y) Z))
((PT X Y Z)
  (PT (X) Y Z))
```

relația "PT" realizează parțial o simplificare a termenilor cu exponent

D) dacă X este un produs de 2 subliste X = Z * x atunci derivata X = (derivata Z) * x + Z * (derivata x) = Y

```
((DRV X Y)
    (CZ * x X)
    (DRV Z y)
    (DRV x z)
    (TMS y x X1)
    (TMS Z z Y1)
    (SM X1 + Y1 Y))
```

relația "C" determină Z și x
derivata Z este y
derivata x este z
y * x = X1
Z * z = Y1
X1 + Y1 = Y = derivata X

E) dacă X este un raport de 2 subliste X = Z / x atunci derivata X = = ((derivata Z) * x − Z * (derivata x)) / (x ↑ 2)

```
((DRV X Y)
    (CZ / x X)
    (DRV Z y)
    (DRV x z)
    (TMS y x X1)
    (TMS Z z Y1)
    (SM X1 "−" Y1 Z1)
    (PT x (2) x1)
    (TMS Z1 (/ | x1) Y))
```

relația "C" determină Z și x
derivata Z este y
derivata x este z
(derivata Z) * x = X1
Z * (derivata x) = Y1
(derivata Z) * x — Z * (derivata x) = X1 — Y1 = Z1
(x ↑ 2) = x1
Z1 / x1 = Y = derivata X

F) derivata (X ↑ 0) este 0

```
((DRV (X ↑ Y) (0))
  (N-EQ Y (0)))
```

G) derivata (X ↑ Y) este 0 dacă X = 0 sau X = 1

```
((DRV (X ↑ Y) (0))
   (OR ((N-EQ X (0)))
      ((N-EQ X (1)))))
```

H) derivata t este 1 (t este variabila de derivare)

```
((DRY t (1)))
```

I) derivata (t ↑ X) = X * (t ↑ (X — 1))
dacă X este constantă (nu este funcție de t)

```
((DRV (t ↑ X) Y)
   (CLLST X Z)
   (NOT IN Z t)
   (SM Z "-" (1) x)
   (PT (t)  x y)
   (TMS Z y Y))
```

relația "CLLST" formează lista Z din X
relația "NOT IN" verifică că Z este o constantă (nu depinde de variabila t)
relația "SM" decrementează exponentul cu 1
relația "PT" plasează exponentul
relația "TMS" generează rezultatul final

J) derivata (X ↑ Y) = Y * (derivata X) + (X ↑ (Y — 1))
dacă Y este o constantă

```
((DRV (X ↑ Y) Z)
   (CLLST Y x)
   (NOT IN x t)
   (CLLST X y)
   (SM x "-" (1) y)
   (PT y z X1)
   (DRV y Y1)
   (TMS Y1 X1 Z1)
   (TMS x Z1 Z))
```

relația "CLLST" închide lista Y
relația "NOT IN" verifică : Y este constantă
relația "CLLST" închide lista X
relația "SM" decrementează exponentul cu 1 : $y = Y - 1$
relația "PT" construiește $X \uparrow (Y - 1)$
relația "DRV" calculează derivata X
relația "TMS" construiește (derivata X) $* (X \uparrow (Y - 1))$
relația "TMS" generează rezultatul final.

K) derivata $(X \uparrow Y) =$ (derivata Y) $*$ (ln X) $* (X \uparrow Y)$
dacă X este o constantă

```
((DRV (X ↑ Y) Z)
 (NOT IN (X) t)
 (CLLST Y x)
 (TMS ((ln X)) (X ↑ Y) y)
 (DRV x z)
 (TMS z y Z))
```

relația "NOT IN" verifică că X este o constantă
relația "CLLST" construiește lista Y
relația "TMS" construiește (ln X) $* (X \uparrow Y) = y$
relația "DRV" calculează derivata $Y = z$
relația "TMS" generează rezultatul final

L) $(X \uparrow Y)$ cazul general : X și Y funcție de t
derivata $(X \uparrow Y) =$
$=$ ((derivata Y) $*$ (ln X) $+ Y *$ (derivata X) $/ X) * (X \uparrow Y)$

```
((DRV (X ↑ Y) Z)
 (CLLST X x)
 (DRV x y)
 (CLLST Y z)
 (DRV z X1)
 (TMS X1 ((ln X)) Y1)
 (TMS y (|/ X) Z1)
 (TMS z Z1 x1)
 (SM Y1 + x1 y1)
 (TMS y1 (X ↑ Y) Z))
```

relația "CLLST" închide lista X
relația "DRV" calculează (derivata X) $= y$
relația "CLLST" închide lista Y
relația "DRV" calculează (derivata Y) $=$ X1
relația "TMS" formează lista : (derivata Y) $*$ (ln X)
relația "TMS" formează lista : (derivata X) $/$ X
relația "TMS" formează lista : $Y *$ (derivata X) $/$ X

relaţia "SM" formează lista:
(derivata Y) * (l n X) + Y * (derivata X) / X
relaţia "TMS" generează rezultatul final.

Observaţie: relaţiile "CLLST" sînt necesare deoarece în lista—funcţie—de—derivat nu se folosesc parantezele decît pentru ridicarea nedeterminărilor (în scopul simplificării scrierii ei).

M) derivatele funcţiilor matematice standard: exponenţiala, logaritm natural; sinus; cosinus; tangenta; cotangenta; arc sinus; arc cosinus; arc tangent; arc cotangent; sinus hiperbolic; cosinus hiperbolic; tangenta hiperbolică.

```
((DRV (exp t) ((exp t)))
((DRV (ln t) (1/t)))
((DRV (sin t) (cos t)))
((DRV (cos t) ("-" sin t)))
((DRV (tg t) (1/(cos t) ↑ 2)))
((DRV (ctg t) (-1/(sin t) ↑ 2)))
((DRV (arcsin t)
    (1/(1 "-"t ↑ 2) ↑ 5. 0 E - 1)))
((DRV (arc cos t)
    (-1/(1 "-" t ↑ 2) ↑ 5 . 0E - 1)))
((DRY (arctg t)
    (1/(1 + t ↑ 2))))
((DRV (arc ctg t)
    (-1/(1 + t ↑ 2))))
((DRV (sh t) (ch t)))
((DRV (ch t) (sh t)))
((DRV (th t) (1/(ch t) ↑ 2)))
```

N) derivarea funcţiilor compuse: X (Y (t))
derivata (X (Y (t))) =
= (derivata X ca funcţie de t dar de variabila Y) * (derivata Y)

```
((DRV ( X Y ) Z )
    ( F X)
    ( D R V ( X t) x)
    ( D R V Y Y)
    ( P U T Y IN x z)
    ( TMS  y z Z))
```

relaţia "F" verifică X este o funcţie simplă (matematică standard)

relaţia "DRV" calculează derivata X (t) = x
relaţia "DRV" calculează derivata Y = y
relaţia "PUT" înlocuieşte în derivata X pe t cu Y
relaţia "TMS" generează rezultatul final

1) Definirea relaţiei "F"

```
((F exp))
((F ln))
((F sin))
((F cos))
((F tg))
((F ctg))
((F arcsin))
((F arccos))
((F arctg))
((F arcctg))
((F sh))
((F ch))
((F th))
```

2) Definirea relaţiei "PUT"

a) dacă funcţia este t şi trebuie înlocuit cu X atunci rezultatul este X

```
((PUT X IN t X)
    /)
```

b) dacă funcţia este o constantă Y în care nu apare t

```
((PUT X IN Y Y)
    (NOT LST Y)
    /)
```

c) cazul general

```
((PUT X IN Y Z)
    (ISALL x y
      (IN Y z)
      (PUT X IN z y)
      (NOT EQ y ()))
    (INVL x Z))
```

relaţia "ISALL" construieşte lista—rezultat Z în ordine inversă = x
relaţia "IN" caută toate elementele z ale listei Y în care se face înlocuirea

relaţia "PUT" face înlocuirile în z şi obţine y dacă y nu este lista vidă

relaţia "INVL" restaurează ordinea.

Primitiva PROLOG "/" atenţionează că numai una din aceste reguli se poate folosi.

O) interfaţa cu utilizatorul

Utilizatorul poate extinde setul de funcţii derivabile cu funcţii noi eventual inexprimabile prin combinaţiile funcţiilor standard. Ulterior, după extinderea bazei de date, noile funcţii pot fi utilizate în noi expresii.

```
((DRV X Y)
  (GET X Y))
```

1) Definirea relaţiei "GET"

a) dacă X este o funcţie simplă

```
((GET (X t) Y)
 /
 (GET - IS (X t))
 (GET-TOLD (X t) Y)
 (ADDCL ((F X)) 0))
```

relaţia "GET—IS" interoghează utilizatorul dacă expresia (X t) are sens

relaţia "GET—TOLD" cere utilizatorului derivata expresiei (X t) care este Y

primitiva "ADDCL" adaugă la baza de date ((F X)) pe primul loc (0) pentru utilizări ulterioare

b) cazul general, funcţii compuse

```
((GET X Y)
  (GET-IS X)
  (GET-TOLD X Y))
```

2) Definirea relaţiei "GET—IS"

```
((GET-IS X)
      (P P Are sens :)
      (P P X ?)
      (R Y)
      (OPLST Y Z)
      (R E x)
      /
      (IN x Z)
  ((R E (D d DA Da da)))
```

3) Definirea relaţiei "GET–TOLD"

```
( ( GET – TOLD X Y)
  /
  ( PP Care este derivata :)
  ( PP X ? )
  ( R Y )
  ( V Y )
  ( A DDCL ( (DRV X Y) ) 3 2 7 6 7) )
```

Observaţie :

1) ((F X)) se introduce la început pentru ca la o utilizare ulterioară să nu se caute în toată lista
2) ((DRV X Y)) trebuie introdusă ultima.
3) În cazul funcţiilor compuse mari pentru care la derivare nu există spaţiu suficient pe stiva, acestea se pot descompune în 2 sau mai multe subfuncţii prin utilizarea ultimei reguli "DRV".

Programul complet se obţine prin concatenarea tuturor regulilor definite, în ordinea în care au fost introduse

BIBLIOGRAFIE

* * * **PC – Magazin :** 1/1990, 2/1990, 3/1990, 4/1990, 1/1991, 2/1991, 3/1991

* * * **If :** 7/1991 8/1991, 1/1992, 2/1992

M. Becheanu, A. Dincă şi al. – *Algebră pentru perfecţionarea profesorilor.* EDP, Bucureşti, 1983

J. R. Bunals – *Begining Micro PROLOG*, Ed. Harper & Row Publishers, London 1983

K. L. Clark, J. R. Ennals, F. G. McCabe – *A Micro PROLOG PRIMER*, Ed. L.P.A., London 1981

W. F. Clocksin, C. S. Mellish – *Programming in PROLOG*, Springer Verlag, Berlin, 1984

TABLA DE MATERII

Lei 156

PROLOG — limbaj de programare al sistemelor expert, pentru calculatoarele de generaţia a V-a după cum afirmă experţi japonezi la aceste proiecte. Prezenta carte realizează o introducere în acest limbaj al inteligenţei artificiale, introducere orientată în special pe Micro-PROLOG disponibil pe calculatoarele compatibile cu ZX Spectrum (HC 85, HC 90, Cobra, TIM-S, Cip, Jet), precum şi sub sistemul de operare CP/M. Sînt definite multe aplicaţii (relaţii) de uz general, din domeniul matematicii şi de analiză gramaticală.

În curs de apariţie:

— **GHIDUL utilizatorului SPECTRUM**
Conţine scheme hard, harta memoriei ROM, modurile de utilizare ale tuturor compilatoarelor disponibile: BASIC, BETA-BASIC, FIFTH, Asamblor, Dezasamblor, PASCAL, C, FORTH, PROLOG.

* * *

În curs de apariţie:
M. M. Popovici — BASIC pentru calculatoarele ZX SPECTRUM, HC, TIM-S, COBRA, CIP, JET, . . .

- **Instrucţiuni — Exerciţii — Probleme**
GHID complet de iniţiere şi conducere în programarea calculatoarelor.
- **Colecţie de programe**
Programe tehnico-ştiinţifice, de matematică, de interes general, de divertisment (jocuri), precum şi programele BETA BASIC şi HISOFT BASIC.

— **LIMBAJUL MAŞINĂ al calculatoarelor ZX SPECTRUM, HC' TIM-S, COBRA, CIP, JET, . . .**
Limbajul de asamblare Z80, ilustrat cu peste 150 rutine care realizează spectaculoase efecte vizuale, sonore, de scriere, de animaţie, etc.

*

ISBN 973 — 95175 — 5 — 2

EDITURA APH — SRL